REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1994	

4. TITLE AND SUBTITLE

Software Reengineering Risks Taxonomy Report

5. FUNDING NUMBERS

6. AUTHOR(S)

Authors: Ms. Joan Smith, DISA/JIEO/CFSW
Mr. Shawn Bohner, The MITRE Corporation

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Defense Information Systems Agency
JIEO/Center for Software, Code TXE
701 South Courthouse Road
Arlington, VA 22204-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

same as above.

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Field 25 should contain the identifier CIM (Collection), as detailed in A. Washington DTIC-OCS IOM, dated April 11, 1994.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document describes a comprehensive taxonomy of software reengineering risks for clarifying software reengineering decisions for automated information systems. The taxonomy is designed to assist program and project managers in the Department of Defense with identification of risks in their efforts to modernize automated information systems.

14. SUBJECT TERMS

Subject Terms: software reengineering risk, software reengineering risk identification, project planning, reengineering, reverse engineering.

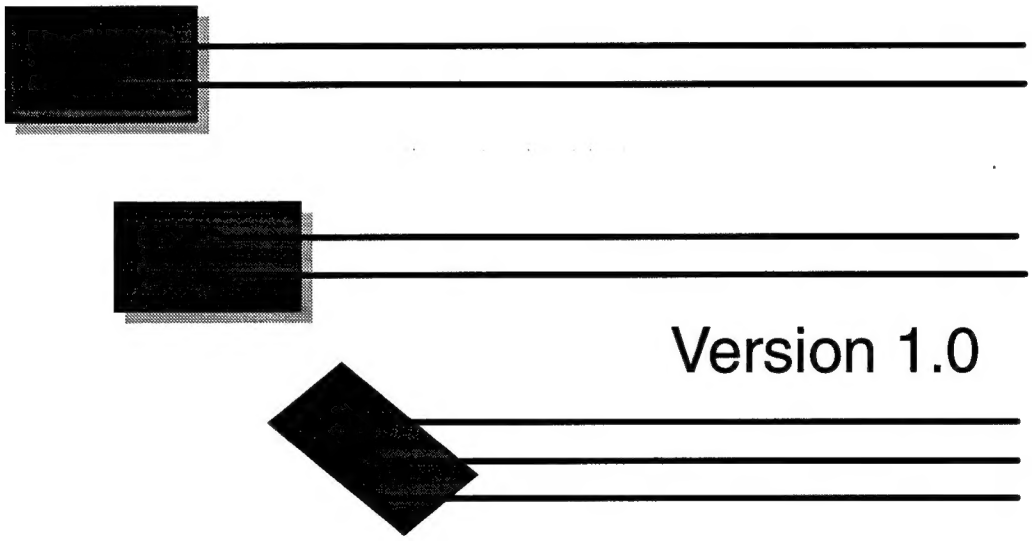
15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	

Defense Information Systems Agency
Joint Interoperability and Engineering Organization
Center for Software
701 South Courthouse Road; Arlington, VA 22204-2199

Center for Information Management Automated Information Systems Software Reengineering Risks Taxonomy Report



Version 1.0

Prepared by:

Software Systems Engineering Department
Software Engineering Technology Division

FOREWARD

The Center for Information Management is now the Center for Software. It remains a part of the Joint Interoperability and Engineering Organization, Defense Information Systems Agency. Since CIM Software Reengineering Risks Taxonomy was presented at the 6th Annual Software Technology Conference, 1994, the title and internal references to the proponent organization remain as presented.

The Center For Software, Software Systems Engineering Department, Software Reengineering Program is located at 5600 Columbia Pike, Falls Church, VA, 22041.

ACKNOWLEDGEMENT

The Automated Information Systems Software Reengineering Risks Taxonomy was prepared by Shawn A. Bohner, Stan Hopkins, and Steve Miller of the Mitre Corporation in coordination with the Defense Information Systems Agency, Center for Information Management, Software Systems Engineering Directorate, Reengineering Division (TXER). Subsequent editorial comments have been added to the original document by Joan G. Smith, TXER. Ms. Smith wishes to thank Cynthia Wright and Tamra Moore of TXER, Paul Johson and Lt. Colonel Dennis Bowers, USAF, of the Software Systems Engineering Directorate for their reviews and comment which significantly contributed to the development of the Taxonomy.

EXECUTIVE SUMMARY

The Department of Defense (DoD) Corporate Information Management (CIM) program concentrates on reducing non-value-added work and costs originally highlighted in the Secretary's of Defense Management Report (DMR) to the President. In support of CIM, the Defense Information Systems Agency, Joint Interoperability and Engineering Organization, Center for Information Management (DISA/JIEO/CIM) is chartered to provide information management technical services to the DoD community. The DISA/JIEO/CIM Software Systems Engineering Directorate, Reengineering Division is responsible for assessing and promoting current reengineering technology in DoD modernization efforts.

This report is a result of joint effort by CIM and Mitre. It represents a comprehensive taxonomy of reengineering risks for clarifying Automated Information Systems (AIS) software reengineering decisions. The taxonomy is designed to assist program and project managers with identification of risks in the AIS modernization efforts.

The report includes: an introduction, an overview of software reengineering, the software reengineering risks taxonomy, an example on using the taxonomy, and a conclusion. The introduction presents background information on the reengineering risks problem, outlines the scope and audience of the report, and defines risk, software reengineering, and software maintenance terms. The overview of the software reengineering addresses key points in reengineering technology and current software reengineering techniques included in the software reengineering taxonomy. The software reengineering risks taxonomy section describes the key points of the taxonomy and presents tables for each of the five key software reengineering risk areas: Planning, Process, Personnel, Product, and Technology. The U.S. Air Force's Weighted Airman Promotion System (WAPS) reengineering project, which was part of the fiscal year 1993 tasking, is presented as an example use of the taxonomy. The report concludes with some comments on the taxonomy and its potential use.

There are a number of techniques that fall under the term reengineering including redocumentation, restructuring, reverse engineering, forward engineering, and translation. This report includes definitions and a brief discussion of these topics, along with perspectives on software reengineering as a whole. Additionally, reengineering for reuse and migration are described as composite techniques that consist of both reverse engineering and forward engineering for specific purposes.

There is not a great deal of experiential data to draw upon for identifying risks due to the relative immaturity of the software reengineering discipline. Two key elements of risk considered in this report are impact and exposure. The report focuses on internal risk (risk associated with the product and its production) rather than external risk (those associated with the failure of the product). The software reengineering risks taxonomy concentrates on risk identification, an essential activity for effective risk assessment, risk analysis, and risk management.

A limited number of published works that directly address software reengineering risks are available. However, after reviewing reengineering and risk literature, interviewing information systems personnel, and brainstorming with staff experienced in software reengineering, many software reengineering risk issues were identified. A synthesis and organization of the collected risk information led to the classification of risk areas by the five major software reengineering risk categories. The taxonomy is by no means exhaustive but does constitute a relatively comprehensive set of software reengineering risks.

The taxonomy provides a means to identify, classify,and organize software reengineering risks to assist project managers and staff involved in AIS software reengineering in understanding their reengineering situations. It simplifies the task of identifying reengineering risks and assists in preparing for estimation and evaluation of software reengineering risks. Table ES-1 summarizes the contents and organization of the software reengineering risks taxonomy detailed in this report.

Table ES-1. Summary of Software Reengineering Risks Taxonomy

Risk Category	**Risk Areas**
Planning	Reengineering Strategy Reengineering Approach Reengineering Goals
Process	Preparation Activities Reverse Engineering Forward Engineering
Personnel	Availability Knowledge - Experience/Training Motivation Teaming
Product	Application Characteristics Technical Infrastructure Implications Standards Characteristics of Data Enterprise Model Implications Age Intended Longevity Importance to Organization
Technology	Methods Tools

Note: In this report, an enterprise model is defined as a high-level model of an organization's information resources documented by an enterprise's data subject areas (entity types or classes), functions, processes, and their inter-relationships. An enterprise model provides guidance to information systems planning and can facilitate reengineering by providing focus and key areas to consider. Detailed activity and

data models are used during business process reengineering and are reconciled with the organization's enterprise model. If an organization does not have an enterprise model a risk of not selecting appropriate information systems is present when performing reengineering. An enterprise model is a key factor when establishing an integrated, shared data environment and should be considered when planning for reengineering.

The Risks Taxonomy identifies risks associated with the software reengineering process defined in the CIM Software Systems Reengineering Process Model (DISA/CIM 1993). The Model is in IDEF0 format which consists of activities and interfaces.

The planning for a software reengineering project consists of selecting a reengineering strategy and appropriate approaches to achieve identified strategic reengineering goals. The risks identified in the planning category are associated with the "Define Project" activity in the *CIM Software Systems Reengineering Process Model* (DISA/CIM 1993). Impacts of planning risks are described in terms of effort, schedule, and costs. Poor planning is a major risk issue for a system project and has an increased emphasis during reengineering due to the level and amount of change associated with the project

The reengineering process risks category identifies risks associated with the critical software reengineering process activities. The risks identified in the process risk category impact the majority of the activities in the *CIM Software Systems Reengineering Process Model.* The primary software reengineering process risk areas in this category are *Preparation Activities*, *Reverse Engineering,* and *Forward Engineering*. By not establishing, organizing, and executing the process effectively, the reengineering effort is exposed to the risk of considerable delays (false starts, redundant activities, etc.) and greatly influences the quality of the resulting product.

Personnel risk issues are critical in software reengineering efforts. The risks identified in the personnel risks category impact the "Project Team" and "Allocate Resources" activities as defined in the *CIM Software Systems Reengineering Process Model.* Reengineering staff availability, knowledge, motivation, and teaming all play key roles in the success of a software reengineering project. The impact of not having available, knowledgeable, and motivated staff in the right teaming arrangement is a software reengineering project that suffers low productivity, poor quality products, and potentially dissatisfaction with the resulting system.

Characteristics of the product influence the potential exposure to software reengineering risks. The risks identified in the product risks category impact the majority of the interfaces in the *CIM Software Systems Reengineering Process Model.* The difference between the existing and target product characteristics is a primary driver is reengineering effort and costs. Relatively minor changes in the product are likely to be less risk extensive when compared to projects with many substantial changes in the product. Other implications that involve the product, such as its importance to the organization and its support to the enterprise as a whole, may also introduce reengineering risks.

Software reengineering technology risks are among the most misunderstood of the risks detailed in this taxonomy. The risks identified in the technology risk category impact he "Identify Methodologies and Tools" activity, and also "Available Reengineering Technology",

"Methodologies", and "Tools" as defined in the *CIM Software Systems Reengineering Process Model.* Technology can be effective in expediting a software reengineering project, but when relied upon inappropriately it can expose the project to considerable additional risk. The management of expectations surrounding reengineering tools is critical, especially in today's climate of producing integrated tools designed for all situations. If expectations are not managed, the impact on time, effort, and costs can be devastating.

The demonstration of the use of a software reengineering risks taxonomy is included as an example of a recently reengineering effort for the United States Air Force. Table ES-2 summarizes the example of using the Software Reengineering Risks Taxonomy. The Weighted Airman Promotion System (WAPS) was a twenty-year-old COBOL system running on proprietary hardware. The redesigned WAPS is to run on an AT&T 3B2 UNIX machine, incorporating relational database technology, with Ada code.

Table ES-2. Example Use of Software Reengineering Risks Taxonomy

Planning Risk Degree: High	The WAPS reengineering project was executed as a complete redesign that pursued multiple disparate goals and objectives. The amount and nature of the change attempted by the project introduced a high degree of risk to the planning risk areas.
Process Risk Degree: Medium/ High	The software reengineering process used during the WAPS project was initially extensively a manual process and focused on data and database conversion issues. The reverse engineering and forward engineering activities were able to take advantage of CASE tools to automate part of the process. The process risk areas of the WAPS project were exposed to a medium to high level of risk due to the manual nature of the majority of the process and the relatively new technologies used.
Personnel Risk Degree: Low	The main risk issues associated with the personnel during a reengineering project: availability, knowledge, experience, training, and teaming; were mitigated by the selection and organization of the WAPS project team. This risk category was of a relatively low exposure during the WAPS project.
Product Risk Degree: High	The WAPS main product related risks areas identified by the taxonomy were in the application characteristics, technical infrastructure, and standards areas. The WAPS reengineering project converted a medium sized legacy system to a new platform using minimal system services. The documentation was converted to a standard (DoD-2167A) format and the data architecture was significantly altered.
Technology Risk Degree: Low/Medium	The technology used by the WAPS reengineering project was determined by the overall goals of the project and the selected approach. The WAPS project was a reengineering effort that migrated the application to a new platform. The WAPS project was a data centered project that was relatively well supported by tools and therefore subject to somewhat less risk.

Although the risk was high for this example, excellent mitigation strategies facilitated risk management and control. For example, a data reengineering technical approach was employed based on the amount of code that would be replaced by the relational database system. The WAPS redesign project was a success because of the attention paid to risks, risk mitigation strategies, and highly motivated and professional staff. This type of success is more likely when software reengineering risks are identified early and risk avoidance plans are put in place.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 1

INTRODUCTION

1.1 PURPOSE

The purpose of this report is to present a relatively comprehensive taxonomy of software reengineering risk issues for clarifying Automated Information System (AIS) software reengineering decisions. This report assists program and project managers involved in AIS modernization efforts in identifying potential software reengineering risks and posturing to avoid them.

1.2 BACKGROUND

Software reengineering is increasingly becoming an important capability as information systems development and maintenance organizations attempt to contain maintenance costs, preserve investments in their software assets, and modernize their information systems (Ulrich 1991). Enormous investments in Department of Defense (DoD) AIS have created formidable maintenance and evolution liabilities that the charter of Defense Information Systems Agency (DISA) addresses. Even with shrinking software budgets, the DoD AISs are in more demand today than ever before. Yet current AISs are aging and becoming outdated. Costs of maintaining these systems escalate to the point of inhibiting their evolution. It is also recognized that many of these systems replicate functions of other AISs in the DoD. Consolidation of these systems appears to have a great potential for reducing costs.

Given the need to evolve AISs, there are a few alternatives for proceeding.

1. Remain on the path of continued *redevelopment* as the business needs of the DoD evolve. Risks of this redevelopment alternative include high costs of developing new systems (often while maintaining old ones) and include the inclination toward stovepipe systems that require high costs to operate and maintain. Even when new development integrates existing systems, many of these risks are present.

2. Patch and extend functionality on existing systems with *continued maintenance*. The risks here include the accelerating costs of maintaining patched-up systems (possibly limited by a finite budget while failing to satisfy essential mission requirements), reduced reliability (perhaps resulting in downtime and safety costs). Ultimately, the second alternative leads to some of the same consequence as the first.

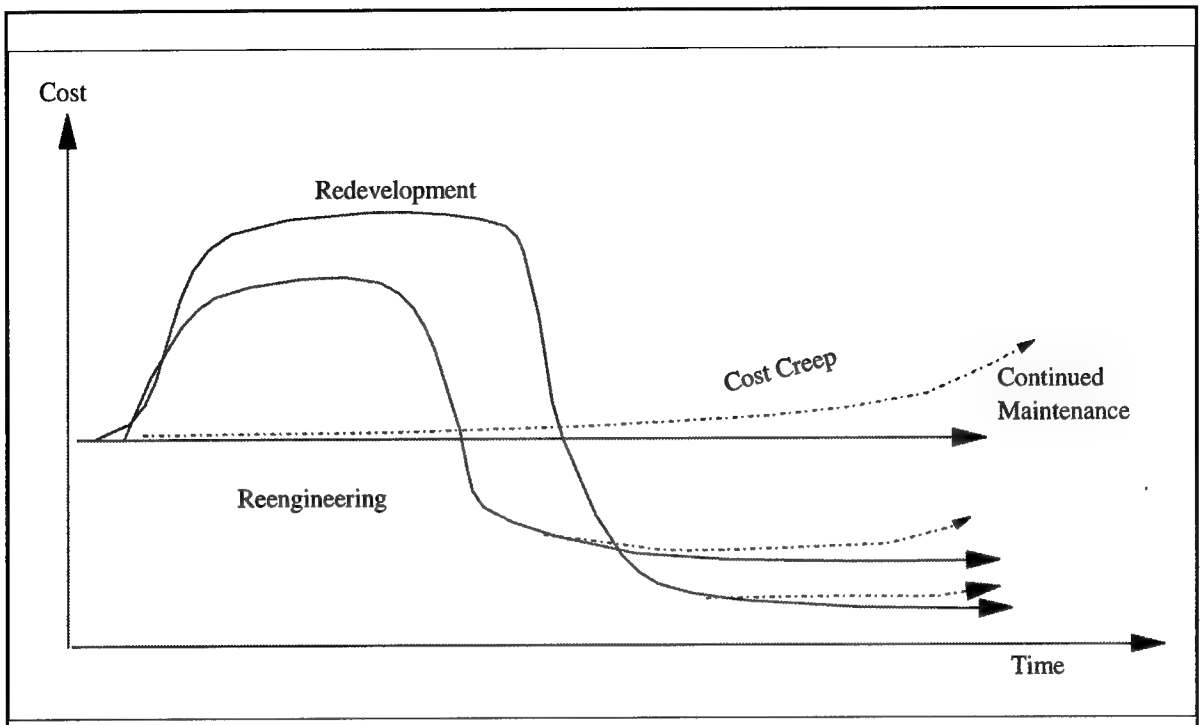


Figure 1. Reengineering Cost/Benefit Over Time

3. *Reengineering* existing AISs is another alternative for addressing needed changes. Software reengineering offers potentially lower risks (and costs) than the first two alternatives, contingent on the characteristics of the effort. Categories of software reengineering risks include planning, process, personnel, product, and technology. Knowing risk issues based on these categories helps clarify the decision between reengineering and the other two alternatives.

Figure 1 illustrates some cost trends over time associated with software redevelopment, continued maintenance, and reengineering. Although the trends in this diagram are not based on any directly measured data, they express the generally accepted experience of industry and government (Bush 1988). The large curve represents redevelopment costs since it is typically expensive and time consuming to redevelop software systems. The smaller projection represents costs associated with software reengineering where it is feasible. The straight line across represents continued maintenance. One example of this type of trend is discussed in a case study by DST Systems, Incorporated (Rochester and Douglas 1991), where it was determined that the cost to redevelop the system would cost $50 million and take several years to complete. Instead, DST Systems analyzed their applications portfolio, identified key areas needing improvements, and reengineered the system for $12 million.

Notice that the software reengineering cost may be smaller and the product may be available sooner, but the level of cost improvement is likely to be less than full redevelopment. This trend is because reengineering an "existing system" often retains the system's inefficiencies over time. Redevelopment is an alternative that will address these inefficiencies and produce a more optimal implementation.

Also notice that each of the trend lines has an associated dotted line that indicates the cost creep of evolving systems. This represents the more likely trend since the software will gradually degrade with maintenance changes once it has been placed in operation (unless proactively controlled) (Lehman, 1980; Horowitz, 1991; Sneed, 1991). Cost creep is an accelerating curve and is thus more pronounced for the continued maintenance alternative since it enters maintenance sooner than reengineering or redevelopment. All three situations are subject to cost creep unless the software maintenance characteristics are proactively controlled (Bohner 1992).

The need to enhance functionality, to improve efficiency or maintainability, to provide greater interoperability with other systems, to consolidate multiple systems with overlapping functionality, and to migrate towards open systems environments can influence reengineering decisions. To serve those making software reengineering decisions, this report offers a taxonomy outlining critical software reengineering risk issues to consider.

1.3 SCOPE

This report concentrates on presenting software reengineering risks in the context of DoD Automated Information Systems from the DISA/JIEO/CIM data intensive systems perspective. Although this software reengineering taxonomy is relatively comprehensive, it is not exhaustive and does not cover all known risks. This report does not focus on software risks in general nor does it purposely duplicate risks from other fields. Instead, it focuses on risks that directly affect software reengineering efforts.

Recognizing that software reengineering seldom operates to the exclusion of other activities, references to address influential aspects from other reengineering disciplines are included. Since automated information systems support business processes, process implication assumptions from the improvement (incremental change) and innovation (radical change) perspectives are included. There is also recognition of systems engineering implications that potentially influence and are influenced by software reengineering decisions. Although these are sometimes mentioned, they are not an emphasis of this work.

The intended audience is DISA and Central Design Activity (CDA) staff and project managers involved in AIS software reengineering or obtaining software reengineering services. Much of the emphasis is on legacy systems from the software maintenance

perspective. With the DoD's intent to consolidate large information systems, there is an emphasis on legacy systems, conversion, and migration in the software reengineering risks taxonomy.

1.4 Definitions and Terminology

In this part of the introduction we examine some terms and definitions used in the report. To set the reader's perspective on risk identification, we first discuss definitions of risk and other risk terminology. We then briefly outline some definitions of reengineering terms used as a preface to the section on reengineering technology. Finally, some software maintenance terms are described to complete the discussion.

1.4.1 Risk Terms

"Risk" is a term that is examined in many fields for various purposes ranging from analyzing economic decisions to determining medical procedures to engaging in war. To this end, Rowe defines risk as "the potential for realization of unwanted, negative consequences of an event" in his book *Anatomy of Risk* (Rowe 1988). Similarly, DoD MIL-STD-1574 defines risk as a "measure of vulnerability to loss, damage, or injury caused by a dangerous element or factor" (MIL-STD-1574). The Defense Systems Management College (DSMC) defines risk as "the probability of an undesirable event occurring (likelihood) and the severity of the consequence of the occurrence (impact)" (DSMC 1989). These are fine for a general perspective on risk, but are there some software-related issues to consider when examining software reengineering risks.

Charette defines formally software risk as "The triplet $<s_i, l_i, x_i>$, where s_i represents the scenarios of what can go wrong, l_i represents the generic likelihood of the scenarios happening, and x_i represents a measure of the consequences of the '$_i$th' scenario. The set of all such triplets forms the totality of risk to the software development being formed." (Charette 1989). Said another way, risk is a function of exposure to loss, chance of loss, and magnitude of loss. Similarly, Chittister of the Software Engineering Institute's Risk Program defines risk as having two primary factors, "the probability or likelihood that an event will occur and the severity of the adverse effects resulting from its occurrence" (Chittister 1992).

In her paper on software maintenance risks (Sherer 1990), Sherer suggests "Risk is measured as a product of the frequency or likelihood of loss and the magnitude or level of exposure." In this case, software risk is defined as the potential loss due to a failure during a specific time period. Sherer goes further to suggest that there are internal risks (those associated with the product and its production) and external risk (those associated with the failure of the product).

The two key elements of risk that we consider in this report are impact and exposure. The likelihood of an undesirable event occurring is beyond the scope of this report since there is

insufficient data to support probability of occurrence. Additionally, we focus on internal risks and occasionally mention external risks where pertinent. Otherwise, there would be considerable time spent enumerating and detecting the vast number of environmental risks associated with software. External risks are covered in Charette's book, "Software Engineering Risk Analysis and Management" (Charette 1989) and in the DSMC document entitled, "Risk Management, Concepts, and Guidance" (DSMC 1989).

Since the software reengineering risks taxonomy in this report concentrates on risk identification, we now examine where it fits in the context of risk management. According to Charette (Charette 1989), risk management is separate from risk analysis and risk identification is part of risk analysis. According to Boehm's terminology (Boehm 1989), risk management is made up of assessment and control. Risk assessment involves identification, analysis, and prioritization. Risk control involves management planning, resolution, and monitoring. According to the DSMC (DSMC 1989), risk management is organized into the collection of activities supporting planning, assessment, analysis, and control. For DSMC, risk identification falls under risk assessment. It is important to notice that risk identification represents the first step in all of these approaches. This indicates that risk identification is an essential activity to risk assessment, risk analysis, and risk management. Moreover, risk identification must take place before effective risk management can occur.

1.4.2 Software Reengineering Terms

The term "software reengineering" has become widely used to describe anything from a software maintenance change to complete redevelopment with no use of existing software artifacts. Arnold defines software reengineering as "any activity that (1) improves one's knowledge of software, or (2) prepares or improves the software itself, usually for increased maintainability, reusability, or evolvability." (Arnold 1993). Chikofsky and Cross define reengineering as "The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form." (Chikofsky and Cross 1990). The CIM Information Systems Criteria for Applying Software Reengineering (DISA/CIM 1993a) extends this to include "The process encompasses a combination of other processes such as reverse engineering, forward engineering, redocumentation, and translation. The goal is to improve the software system (functionality, performance, or implementation)." Attempting a more pragmatic view, we use software reengineering to describe substantial change in major software architectural components of a software system (e.g., source language, operating system, database management system, etc.) or substantial improvements to important attributes of a system (e.g., software maintainability, portability, quality, etc.).

There are a number of techniques that fall under the term reengineering including redocumentation, restructuring, reverse engineering, forward engineering, translation (also known as conversion), reengineering for reuse, and migration. The first five terms outlined below according to Chikofsky et al (Chikofsky and Cross 1990) and defined in the CIM Information Systems Criteria for Applying Software Reengineering (DISA/CIM1993a) are covered in more detail in Section 2.

Redocumentation "The creation or revision of a semantically equivalent representation within the same relative abstraction level. The resulting forms of representation are usually considered alternative views intended for a human audience." Redocumentation uses static analysis of source code to generate information for aiding software maintainers in understanding the software.

Restructuring "The transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior." More pragmatically, restructuring is the automated translation of an unstructured program or other software artifact into its functionally equivalent representation in a structured form.

Reverse engineering "The process of analyzing a subject system to identify the system's components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction." Reverse engineering recreates the specification and/or design information from source code and other software artifacts.

Forward engineering "Within the context of reengineering, forward engineering is the software engineering activities that consume the products of reengineering activities primarily reverse engineering, reuse, and new requirements to produce a target system." After useful software artifacts are captured from the existing system, they are integrated with newly constructed artifacts in the forward progression through the software engineering process.

Translation (also known as Conversion) "Transformation of source code from one language to another or from one version of a language to another version of the same language." Translation is the automated transformation of software artifacts into functionally equivalent artifacts with a different base (e.g., language, database, or user interface).

The last two techniques, reengineering for reuse and migration, make use of the other techniques. Reengineering for reuse (also known as software salvaging and reclamation) captures, selects, adapts, classifies, documents, and stores software artifacts from an existing system to be reused in other software systems (Biggerstaff 1989) (Arnold 1990) (Garnett and Mariani 1990). Migration first reverse engineers functions and data from one or more existing systems, combines and partitions them for allocation, then forward engineers the allocated functions into one or more respective systems (Medek and Boylan 1992).

1.4.3 Software Maintenance Terms

Traditionally, software maintenance is defined as the collection of activities that support correcting, adapting, and perfecting existing software systems (Lientz and Swanson 1980). Corrective maintenance is the reactive correction of software performance and implementation

errors. Adaptive maintenance is software change in response to new requirements for the purposes of enhancing the software. Perfective maintenance is proactive software improvement through performance optimization and quality (maintainability) support for future evolution of the system. Additionally, user support represents the work that a maintainer does responding to user requests that do not result in product modification.

The traditional software life cycle depicts software maintenance as starting after software is deployed. However, software maintenance begins with user requirements, and principles of good software development apply across both the development and maintenance processes. Another way to view the software life cycle is software development followed by multiple instances of software evolution. One of the goals of software development is the production of *maintainable* software systems. By the same token, evolution of software systems should strive to preserve or restore this maintainability. Reengineering is a consideration as an improvement measure prior to system redevelopment when maintainability wanes to the point that changes are excessively difficult. The next section presents an overview of software reengineering by presenting some perspectives on the technology and introducing relevant software reengineering techniques.

1.5 Organization of the Report

The organization of this report is as follows: an introduction, an overview of software reengineering, the software reengineering risks taxonomy, an example on using the taxonomy, and a conclusion. Section 2 presents an overview of software reengineering that addresses key points in the technology and software reengineering techniques included in the software reengineering risks taxonomy. Section 3 outlines the software reengineering risks taxonomy explaining the organization and salient features. It then continues by presenting the tables for the five key software reengineering risk areas: planning, process, personnel, product, and technology. Section 4 uses the taxonomy in an example based on the Air Force's Weighted Airman Promotion System (WAPS) reengineered earlier this year. The report concludes with some comments on the taxonomy and its potential use.

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 2

OVERVIEW OF SOFTWARE REENGINEERING

There is some truth to the statement "you must first engineer a system before you can reengineer it." Many current software systems are not well engineered which makes it difficult to reengineer. Even if these systems are engineered well, they are likely to suffer from continued change associated with software evolution and maintenance (Lehman 1980). Changes in the application environment and concomitant changes to the software product usually lead to software reengineering. Other reasons for reengineering a software system include:

- Outdated/obsolete information technology
  - Introducing new database technology
  - Need to integrate with new systems
  - Introducing new user interface technology
  - Introducing new communications technology

- Need to combine and migrate stovepipe information systems
  - Rearchitect databases
  - Migrate data to new data architecture

- Deteriorating reliability/increased software failures

- Poor response to change
  - Long change cycles
  - Increasing backlog of change requests

- Waning software maintainability
  - Code/design instability
  - Increasing maintenance costs
  - Increasing size and complexity
  - Poor quality code
  - Need to eliminate obsolete/redundant code

- Difficult to test
  - High test case volatility
  - Long test cycles for short changes

- Performance problems

Often software work products become out-of-date, patched, and difficult to maintain with continued changes (Bohner 1992). Reengineering techniques offer some relief to this

situation by improving the maintainability of the software work products (Bush 1988). In this regard, reengineering can proactively change software to improve maintainability.

Requirements, system descriptions, specifications, architectures, functional designs, detailed designs, source code (in its various forms), test specifications, test procedures, test reports, and verification plans are among the plethora of information that inundates the audit trail of the typical software development effort (Santa Barbara I)(Arnold 1993). While much of this information is generated during the forward progression through the software development life cycle, it is currently impossible to reconstruct all of this information from only the source code of a system. Obtaining software development information from the source code of the system is often difficult. Some essential features may be derivable, but many information gaps will remain. While some derived information will be correct, some may be misleading.

Reengineering software often involves information that can be reconstructed form the source code. The degree to which information can be extracted depends largely on the language used, the database interface, the user interface, interfaces to system services, interfaces to other languages, domain maturity/stability, and the tools available. The information useful to the software maintainer depends on the maintainer's experience level (with the above aspects of the system), the complexity of the change, and the complexity of the system being changed. It is important to note that most third generation computer languages are designed to express solutions to the computer and have little if any capability for expressing requirements of design specifications. Current tools can only capture some evidence of program designs, little evidence of architecture, and very little software and system requirements. Much of the reengineering work is accomplished by the experienced software engineer or systems analyst.

Figure 2 illustrates the general view of software reengineering with reverse engineering and forward engineering.

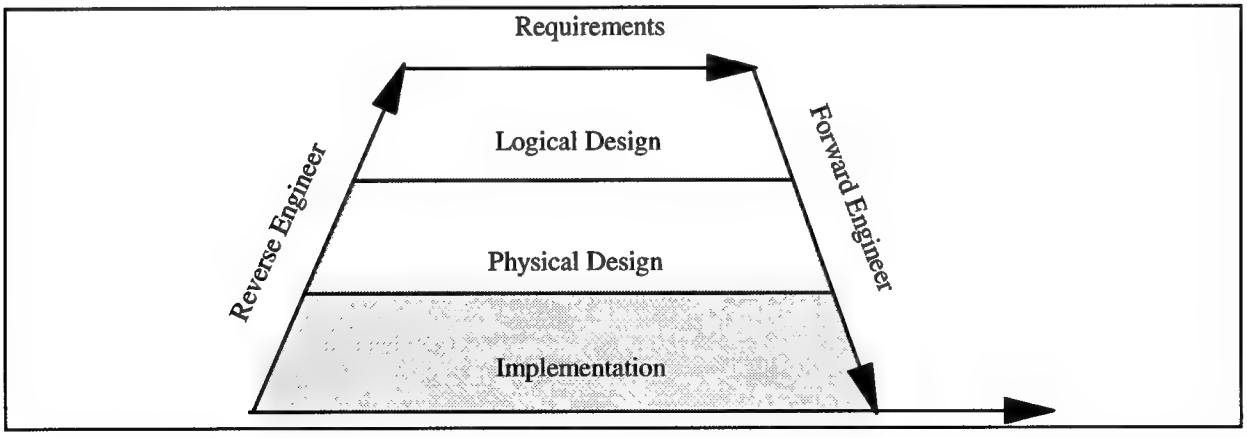
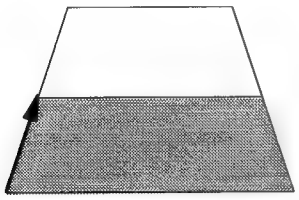


Figure 2. Software Reengineering and Levels of Information

Notice that there are four levels of information represented in the diagram. The first, implementation, represents source code, data definitions, database calls, and other computer interpretable artifacts. This is the most concrete of the levels. The second, physical design, represents program design, physical data model, and other detailed design information. Physical design is where most capture tools have their best success. The third level, logical design, represents high level software and data design specifications (i.e., data flow, control flow, logical data model, entity-relationship-attribute, and the like). This conceptual and logical information about the system is very difficult to capture from implementation or even physical design. Requirements are the most abstract representation portrayed here. Requirements even when formally represented are extremely difficult to derive from lower level artifacts. These are usually captured manually from documentation and interviews with users and systems staff.

This section examines some important reengineering techniques in more detail including: redocumentation, restructuring, reverse engineering, forward engineering, translation (also known as conversion), reengineering for reuse, and migration. As indicated earlier, the first five techniques are singular while the last two are composites of one or more of the other techniques. Each of these reengineering techniques is discussed with respect to the perspective of reverse engineering (or capturing) information to a higher level of abstraction (ranging from implementation to requirements) then forward engineering the system to more concrete levels.

2.1 REDOCUMENTATION



Software Redocumentation

- Captures physical design information
- Does not change implementation or function
- Generates reports on structure and metrics
- Does not store objects in malleable representation

Figure 3. Software Redocumentation

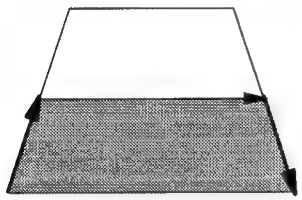
Redocumentation is the analysis of source code to produce software system documentation and metrics information. The analysis measures characteristics of source code to capture variable usage, module calling, control paths, module volume, calling parameters, and/or test paths. Analysis of this information produces stand-alone reports containing useful statistics and diagrammatic information. However, the information produced is not necessarily controlled through a project database/repository or based on software system modeling

methods. The outputs of redocumentation include a wide range of possible documents and reports including:

- Module calling hierarchies
- Hierarchy input process output (HIPO) diagrams
- Data interface tables
- Data flow tables and diagrams
- Control flow tables and diagrams
- Pseudocode
- Test Paths
- Module and variable cross-references

Redocumentation supports software restructuring by providing graphical, textual, and tabular information for the developer to assess whether or not a system requires restructuring. The resulting information form redocumentation significantly aids maintenance efforts by providing module, program, and system information. However, since there is no mapping between specification and restructured code, the resulting documentation reflects what is, rather than what may have been specified in the original documentation.

2.2 RESTRUCTURING



Software Restructuring

- Represents code in internal format
- Alters characteristics internally
- Produces changed implementation without altering function

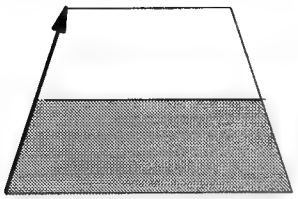
Figure 4. Software Restructuring

Early in its history, restructuring was the primary transformation technology for making source code easier to understand and subsequently easier to maintain. As software engineering principles have played a more important role in software development, restructuring has evolved to become one aspect of what is now called reengineering. "Software restructuring is the modification of software to make it easier to understand, easier to change, and less susceptible to change" (Arnold 1986).

Restructuring tools interpret the source code of a system and represent is internally for restructuring. Simplifications to the internal representations are performed according to transformation rules, and the resulting representations are recast in structured code. The outputs of these tools typically include only source code. However, some restructuring tools have supporting tool suites to provide structure, volume, complexity, and other additional

metric's information. Metrics information determines the maintainability of the source code and the necessity of restructuring. These tools are very useful and have been shown to increase productivity of software maintenance staff substantially.

2.3 REVERSE ENGINEERING



Software Reverse Engineering

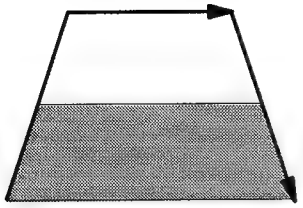
- Captures physical design information
- Infers abstract information (human intensive)
- Generates software documentation
- Stores objects in malleable representation
- Does not change implementation or functionality

Figure 5. Software Reverse Engineering

Reverse engineering, like redocumentation, provides specification and design information about the software system from its source code. However, reverse engineering goes further than redocumentation in attempting to recover engineering information based on software specification and design methods as well as storing it in a form that can be manipulated by a software engineer. The extracted specification and design information is not always complete because the mapping between the source objects and associated design objects is often many-to-many. Therefore, in reverse engineering there is a high potential for information loss in the resulting interpretation.

Recent advances in graphic workstation technology and storage management techniques facilitate reverse engineering automation. The graphics capabilities of workstations allow designs of software to be displayed and manipulated graphically while the storage management capabilities provide a good mechanism for managing a repository of software information gathered by the reverse engineering tool. Source code is first submitted to the reverse engineering tool. This tool interprets the structure and naming information to construct outputs in much the same way as described in the redocumentation tool discussion. Standard analysis and design methods serve as good communication mechanisms to articulate the reverse engineering information. Data dictionaries, data flow, control flow, and entity-relationship-attribute diagrams are all examples of typical reverse engineering outputs.

2.4 FORWARD ENGINEERING



Software Forward Engineering

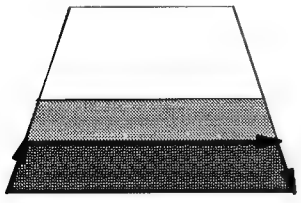
- Integrates captured software artifacts with new ones to construct system
- Generates software documentation
- Stores objects in malleable representation
- Changes implementation and may change function

Figure 6. Software Forward Engineering

Forward engineering represents the software engineering work that accomplishes construction of the reengineering system (the forward progression through the software engineering process from requirements through deployment). In the reengineering context, the key element that distinguishes forward engineering from standard development is the existence of software artifacts captured from the original system during reverse engineering. These artifacts represent effort savings since they do not have to be newly constructed. However, they also increase effort necessary to integrate them with the newly developed software products.

Forward engineering may also introduce new requirements that change functionality of the software being reengineered. Introducing new requirements during the forward engineering phase can complicate the reengineering effort since many of the normal regression tests used for validating the system may be affected. New requirements may substantially impact many of the captured software artifacts and may require additional modification. Examination of new requirements prior to reverse engineering can eliminate conflicts and required rework during forward engineering.

2.5 TRANSLATION



Software Translation

- Captures language or data representation
- Translates to target form (tool intensive)
- Does not change function
- Changes implementation

Figure 7. Software Translation

Software translation (also known as software conversion) is a relatively mature technology applied since the 1960s to one degree or another in the language and data arenas. Translation is much like restructuring in its level of abstraction and its focus on altering the implementation while leaving the function intact. The difference is that translation often changes a key architectural component of the system like implementation language, database, operating system, communication interface, or user interface. Most translation work is accomplished using a tool while exceptions like system services are handled manually. Typical translation issues include representation and structural incompatibilities, coverage, labeling, and mode of control.

2.6 REENGINEERING FOR REUSE

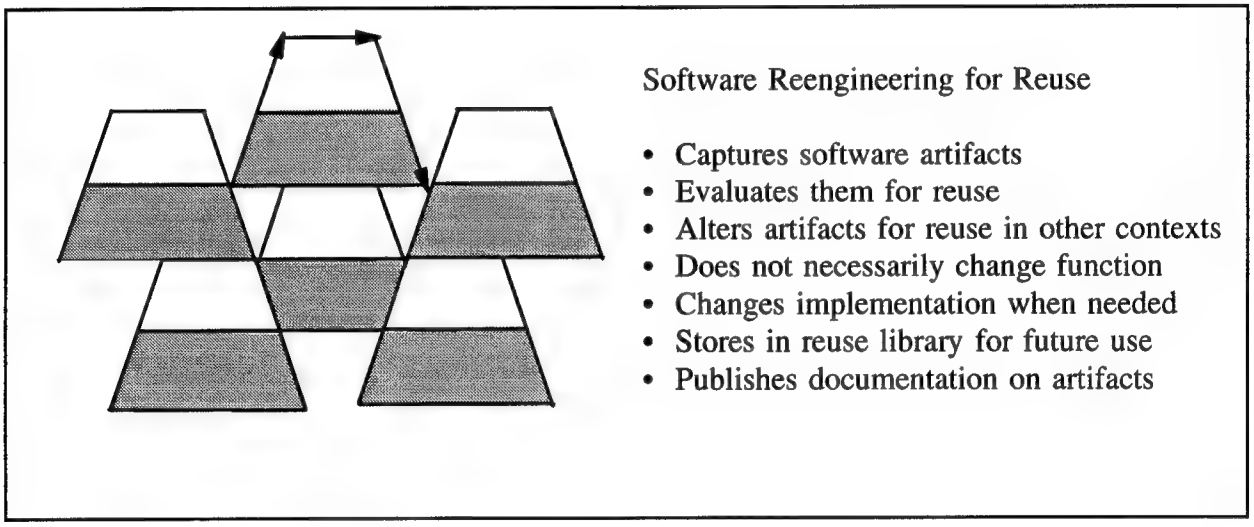


Figure 8. Software Reengineering for Reuse

Software reengineering for reuse (also known as software salvaging and reclamation) is a microcosm of software reengineering in the large. Essentially, this composite technique consists of reverse engineering software components from an existing system and forward engineering them to be placed in a reuse library. Rather than addressing the entire system, the focus is on components that can be reused in multiple applications.

Reengineering for reuse is relatively immature technology that has been applied effectively only since the late 1980s. Reuse, in its current form, is mostly "component reuse" applied to software modules and some design components. Unfortunately, most reuse success has centered on the population of reuse libraries with low level fragments of software programs. Populating these libraries can be accomplished by salvaging software artifacts from existing systems rather that developing them form scratch. Once the artifact has been captured, normal "engineering for reuse" principles apply while forward engineering the artifact.

Capturing artifacts for reuse is not an easy process since there are likely to be considerable numbers of objects that must be evaluated and selected. Once selected, they may need to be modified and will need considerable testing (since a faulty component may inflict the same number of systems as it is reused). Then the reusable component must be documented and published so that users will be aware of its existence and how to use it.

The higher level the artifact, the more gain there is in its reuse but the less likely that it will fit the purposes of a given situation. The converse is true also. Additionally, adaptation of the artifact may represent considerable work both from the capture and use perspectives. If there is considerable work in adaptation, this must be weighed against the artifact's potential for reuse.

2.7 MIGRATION

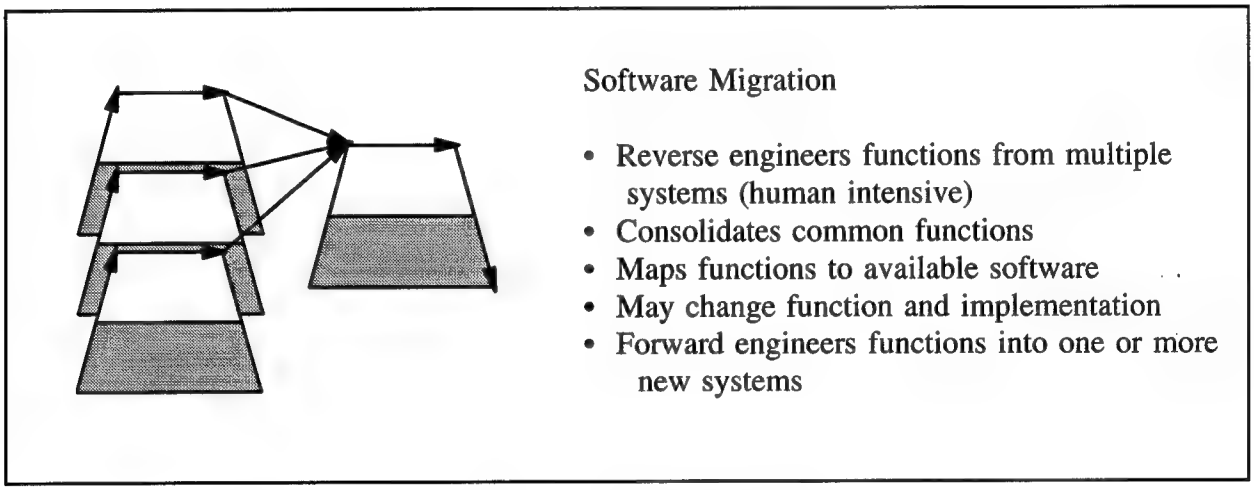


Figure 9. Software Migration

Software migration is another procedure that may apply a composite reengineering technique. If the migration activity is not a simple conversion, such as to a new operating system, it will employ reverse engineering to capture requisite functions and associated data from one or more systems and forward engineers them into one or more new systems. The idea of common function and shared data is key to this technique (DoD 8020.1 1992) (Strassman 1992). Software migration's technology maturity is dependent on the reengineering technologies it employs to accomplish the migration. Migration has (to lesser and greater degrees) been successfully applied to information systems over the last two decades (Davenport 1993). Sometimes the right approach was to get requirements form originating systems and redevelop a single or multiple systems with a remapping of functionality. Other times, common factors such as language would enable entire functions (code and documentation) to be lifted from one or more systems and integrated into common systems. Migration appears to be the technique of choice when moving from a centralized data

approach to a client/server data approach. Therefore, much more experience will be gained with migration over the next few years.

This section presented a brief overview of some current reengineering techniques pertinent to the DoD Automated Information System modernization. In the next section, we examine software reengineering from the perspective of its potential risks. This software reengineering risks taxonomy outlines five risks categories. Many of the reengineering techniques presented here map into the technology risks category. However, many other reengineering issues are discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 3

REENGINEERING RISKS TAXONOMY

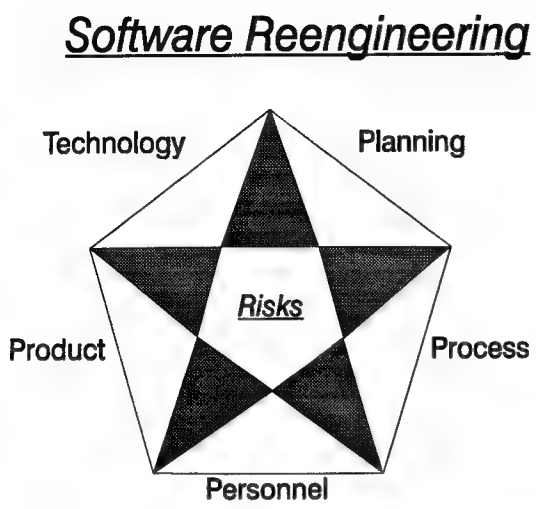
Software reengineering is a relatively new field and does not have a great deal of experiential data to draw upon for identifying risks. There is a limited number of published works that directly address this area (Arnold, 1991; Arnold, 1992; Sneed, 1991). While their work was a beginning, Arnold and Sneed suggest much more work needs to be done. The software reengineering risk taxonomy presented in this section represents a synthesis of over four hundred identified risks affecting reengineering projects. The reengineering risks taxonomy is a product of considerable effort reviewing reengineering and risk literature, interviewing information systems personnel, and brainstorming with staff experienced in software reengineering[1]. The taxonomy is by no means exhaustive but does constitute a relatively comprehensive set of software reengineering risks.

The intent of this taxonomy is to identify, classify, and organize software reengineering risks into a form that will assist DISA and Central Design Activity (CDA) project managers and staff involved in AIS software reengineering situations. Its purpose is to simplify the task of identifying reengineering risks and assist in preparing for estimation and evaluation of software reengineering risks.

In synthesizing and organizing the collected risk information, we identified five major software reengineering risk categories: Planning, Process, Personnel, Product, and Technology. Each one of these categories is divided into several risk areas which are in turn divided into sub-areas. Figure 10 presents the categories in a star configuration to signify the relationship of the categories to each other and that risk can have multiple implications. Each risk category is listed at the right hand side of the diagram in a table that includes the top-level risk areas for each category.

---

[1] The software reengineering risks information was generated from investigations of software reengineering literature much of which is in the list of references at the end of this report. Interviews were conducted with Redesign staff at the Air Forces Military Personnel Center, DISA/CIM staff, GTE Data Services staff in Tampa, Florida, MITRE Staff supporting Internal Revenue Service modernization efforts, and participants of the Reverse Engineering Workshop. The informal list of factors applied to building this taxonomy is not included as part of the report but is available from the authors upon request.



Risk Category	Risk Areas
Planning	Reengineering Strategy Reengineering Approach Reengineering Goals
Process	Preparation Activities Reverse Engineering Forward Engineering
Personnel	Availability Knowledge - Experience/Training Motivation Teaming
Product	Application Characteristics Technical Infrastructure Implications Standards Characteristics of Data Enterprise Model Implications Age Intended Longevity Importance to Organization
Technology	Methods Tools

Figure 10. Five Sides of Software Reengineering Risks

The software reengineering risks taxonomy partitions each of these five risk categories into separate tables. Detailed separate subsections, starting with planning risks, describe the planning, process, personnel, product, and technology risk categories. Each category subsection contains a few paragraphs of introductory text followed by a detailed table outlining risk areas, impact/exposure, and example factors. The table structures present the risk category (e.g., Planning) followed by numbered risk areas (e.g., 3 Reengineering Goals), followed by its risk sub-areas (e.g., 3.1 Maintainability), and so on.

In this report, an enterprise model is defined as a high-level model of an organization's information resources documented by an enterprise's data subject areas (entity types or

classes), functions, processes, and their inter-relationships. An enterprise model provides guidance to information systems planning and can facilitate reengineering by providing focus and key areas to consider. Detailed activity and data models are used during business process reengineering and are reconciled with the organization's enterprise model. If an organization does not have an enterprise model, a risk of not selecting appropriate information systems is present when performing reengineering. An enterprise model is a key factor when establishing an integrated, shared data environment and should be considered when planning for reengineering.

The Risks Taxonomy identifies risks associated with the Software Reengineering Process defined in the *CIM Software Systems Reengineering Process Model.* The Model is in IDEF0 format that consists of activities and interfaces.

3.1 Planning Risk Category

The planning for a software reengineering project consists of selecting a reengineering strategy and appropriate approaches to achieve identified strategic reengineering goals. The risks identified in the Planning risk category are associated with the "Define Project" activity in the *CIM Software Systems Reengineering Process Model.* A reengineering strategy sets an overarching direction for achieving reengineering goals based on strategic organization goals and situational factors, such as department budget or resource availability. Alignment of an approach with the selected strategy provides the means for achievement of reengineering goals. The linking of reengineering goals with overall organization goals is an important factor in reducing the risk associated with reengineering. Alignment of software reengineering goals with organization goals helps to ensure reengineering projects support the organization's strategic direction.

Particular strategies are available for a reengineering project manager to choose to meet an organization's goals. Adoption of a strategy for reengineering provides broad constraints to the tactics or processes executed during reengineering. The intent of a reengineering project's approach is to guide the processes necessary to achieve established reengineering goals. The selected approach should target an appropriate level of abstraction based on reengineering goals defined by the adopted strategy. An organization's budget, time, schedule, adaptability to change, and aversion to risk provide guidance to the selection of a reengineering strategy.

Three common reengineering strategies employed for reengineering projects are innovation, incremental improvement, and tactical strategies. Each of these strategies contains discrete associated risks. An organization's level of aversion to change influences reengineering strategy selection. The strategy selected by an organization for reengineering bounds the available approaches that are suitable for executing a reengineering project plan.

An innovative strategy for reengineering is a radical method that replaces an entire information system during one concentrated reengineering effort. Adoption of an innovative

strategy is a high-risk method that can provide the most benefit if successfully executed. It is analogous to the business process reengineering approaches that advocate complete reinvention of an organization's business processes. An innovative strategy usually dictates a top-down approach that changes an information systems functionality and implementation. The methods associated with this strategy usually have a high risk due to longer schedules and a high degree of change associated with this type of project.

An incremental improvement strategy is a reengineering strategy that replaces the entire information system using a series of phased projects. This strategy guides the organization's reengineering approaches towards less risk intensive projects that are of a shorter duration. While an incremental strategy can be more effective than an innovative strategy in producing short-term benefits it frequently encourages approaches that consume a longer time period to institute all needed changes. The incremental approach introduces less scheduling risk than an innovative approach but may present interface and interoperability issues between the reengineered phase of the system and the existing system.

A tactical strategy for reengineering identifies and selects key areas of an information system to initiate reengineering. A tactical strategy can use either an approach that targets the critical twenty percent of an information system, based on the concept that twenty percent of a system usually accounts for eighty percent of the processing (i.e., the 80/20 principle), or can be specified for specific problem area improvements. A tactical strategy can be viewed as an innovative strategy that focuses on a limited segment of a system.

Multiple approaches are available to implement selected reengineering strategies. One of the following approaches is typically used when reengineering:

- Translation
- Partial change to implementation, no change to function
- Complete change to implementation, no change to function
- Change to implementation minimal, change to function
- Complete change to implementation and function

Example reengineering strategies, approaches, and goals are identified in Table 1 along with more detailed impacts and exposures.

Table 1. Software Reengineering Planning Risks

Planning

In the absence of planning there is considerable risk of having an unfocused or redundant effort. Impacts of planning risks can be described in terms of effort, schedule, and costs. Poor planning is a major risk issue for a system project an has an increased emphasis when reengineering due to the level of change and the number of parameters associated with the project. Selection and management of appropriate strategies, goals, and approaches enable anticipation of potential changes in corporate culture and facilitates maintenance of management commitment.

Risk Area	Impact/Exposure	Example Factors
1 Reengineering Strategy	An effective strategy can have major risks associated with it. Selecting a strategy that is compatible with business process reengineering goals is critical. Innovation can expose the reengineering effort to radical change without commensurate commitment. Incremental improvement can alleviate some of this but can cause the change to be so slow that it is too late to be of value. Tactical strategy is highly dependent on a priori information.	Innovation (radical) Incremental Improvement Tactical (concentrated area) • 80/20 principle • Specific improvement area (e.g., restructure module)
2 Reengineering Approach	Reengineering approach can expose a project to completeness dangers. A top-down approach may make assumptions that miss some of the efficiencies of capturing low level, detailed system information. Bottom-up may overlook the major issues like information technology that the business process may need. Opportunistic approach is again intelligence dependent. Change to the implementation without change to the function of a system is a good separation of concerns. This simplifies the effort and reduces its impact.	Translation Partial change to implementation, no change to function Complete change to implementation, no change to function Change to implementation minimal, change to function Complete change to implementation and function

Risk Area	Impact/Exposure	Example Factors
3 Reengineering Goals	Inappropriate or absent software reengineering goals can lead to substantial exposure to a misguided or unfocused reengineering project resulting in costly rework and delays. If the goals are not agreed upon and measurable, there is exposure to an inability to monitor or control the reengineering activity. Moreover, there is the potential for goals to shift, causing considerable work to satisfy a moving target.	
3.1 Maintainability	Improving information system maintenance is a key success factor for many organizations. By not improving system maintainability the system may be subject to increasing maintenance costs and slower response to change. Difficult to maintain systems have a risk of being by-passed for other solutions as the system becomes increasingly expensive to maintain.	Costs Response to Change Ease of Change Understandability
3.2 Modernization	Modernization of legacy information systems is a common goal for many organizations with large investments in information technology. The number of factors associated with tying system modernization with a reengineering approach increases a project's risk. Introducing new technology into an existing system may mean that considerable segments of the system have to be reengineered, often with cost exposure.	Application Infrastructure Technology Infusion
3.3 Reliability	Reengineering current information systems to improve reliability is a prevalent goal that is linked to other considerations (e.g., platform capabilities)	Reduce Failures Reduce Errors

Risk Area	Impact/Exposure	Example Factors
3.4 Change Functionality	Reengineering is a mechanism for changing an information system's functionality to meet new or changing system requirements. Pursuing modification of a systems functionality by using reengineering introduces several risks, including inappropriate selection of a reengineering approach when a redevelopment would be more effective.	Add New Functions Alter Current Functions Delete Old Functions
3.5 Portability	Reengineering an information system to use open systems technology reduces the risks for future maintenance and redevelopment projects.	Separate Concerns • GUI, DBMS, subsystem Partitioning Simplify Interfaces Standardize
3.6 Incorporation of Standards	Reengineering an information system to incorporate standards may improve the portability and reusability of the system and its components.	Stability of standards, • applicability • extent of acceptance by community at large
3.7 Infrastructure Modernization	The goal to modernize an information system's infrastructure often initiates a reengineering effort. The extent that the target infrastructure deviates from the current infrastructure influences the amount of risk associated with the modernization (e.g., moving from a mainframe based architecture to a client/server environment introduces risk). Exposure of a project to considerable schedule, technology, and cost impacts is probable without careful consideration of the architectural ramifications.	Data management architecture • Centralized to distributed client/server • Hierarchical to relational User interface architecture • Command line interface to graphical • Single mode to multimode windows Communications • Direct connect to Local Area Network (LAN) to Wide Area Network (WAN)

Risk Area	Impact/Exposure	Example Factors
3.8 Repository Load	Populating an organization's repository is a common goal of reengineering projects. The goal of repository population usually involves long-term benefits that introduce risk if an organization is concerned about short-term cost objectives.	Shared repository Repository security Costs and benefits (short and long-term)
3.9 System Integration	Reengineering can integrate existing information systems with newly developed or other reengineered systems. Modernizing existing stovepipe systems with reengineered systems replacing existing data structures with shared data structures represents considerable change and exposure to risk. Moving to a shared data environment frequently includes changing data management technology (e.g. inserting a RDBMS) which can increase overall project risk.	Configuration management • Change control • Configuration control Data management technology

3.2 Process Risk Category

Much of the software reengineering risks associated with the other risk categories have a reengineering process risk component. That is, there are activities that often originate risks that manifest in the other reengineering risk categories. In the absence of appropriate software reengineering process activities, there is considerable exposure to risks.

Reengineering process risks presented here support the Center for Information Management Software Systems Reengineering Process Model (DISA/CIM 1993) by identifying risks associated with critical software reengineering process activities. The primary software reengineering process risk areas are *Preparation Activities, Reverse Engineering,* and *Forward Engineering*. The risks identified in the process risks category impact the Center for Information Management Software Systems Reengineering Process Model's *Define Project, Reverse Engineer,* and *Forward Engineer* activities. Figure 11 depicts a simplified diagram of these activities.

The risks in this category focus on software reengineering process or even normal software activities. Since software reengineering processes differ mostly at the detailed level (based on specific reengineering requirements), the process risk category presents several risk areas that should be examined to ensure that reengineering project staff are cognizant of potential risk exposures. Reengineering *Preparation Activities* include: Establish Goals, Analyze Application Portfolio, Select Reengineering Strategy, Analyze Cost and Benefit of Reengineering Effort, Establish Support for Reengineering Effort, Develop Reengineering

Plans, Select Reengineering Technology, Build Reengineering Team, and Manage Reengineering Acquisition. *Reverse Engineering* activities concentrate on: Capture Objects, Analyze Objects for Reuse, and Prepare Objects for Reuse. *Forward Engineering* activities support: Add new Requirements, Integrating Captured Objects to Construct New System, Reimplement System to Reengineer in the Future, Migrate Existing Data, and Conduct Post-Reengineering Assessment.

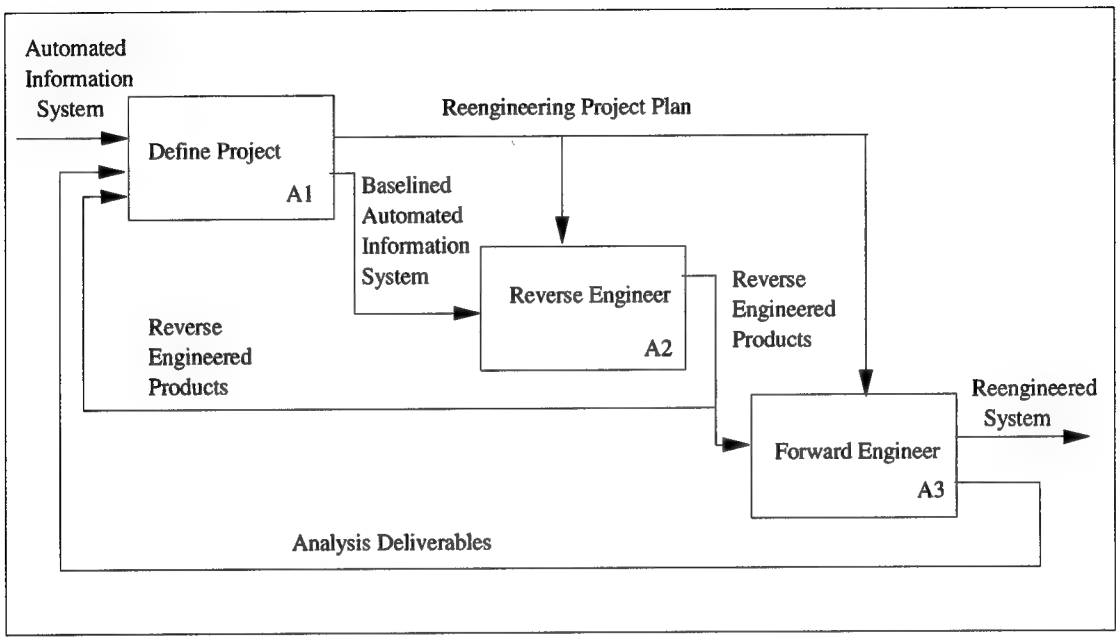


Figure 11. Major Software Systems Reengineering Process Activities

The sub-areas of the Preparation Risks area cause vulnerability to many of the risks presented by the Planning category. During the preparation for a software reengineering project the strategy and goals of the project are established and viable approaches are examined. Determination of both the risks of conducting a reengineering effort and the risks of not conducting the effort is a consideration.

A clear case should be made that a problem exists and can be solved by a reengineering effort before a reengineering project is initiated. From the maintenance improvement perspective, evaluation of the applications portfolio identifies targets before reengineering. The risk of not performing this evaluation is that there may not be the tangible connection between maintenance problems and reengineering solutions. When the goal is to replace several systems with one, there is the risk of not selecting the functional features most representative of the application domain. A need exists for processes to select the migration systems, as well as process steps to introduce the necessary unique requirements of similar legacy systems. Failure to do so risks greater cost and longer schedules for the reengineering

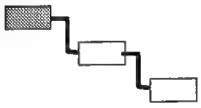
effort as well as the necessity to continue operating the legacy systems after the beginning of operation of the reengineered system.

The Reverse Engineering risk area is typically the riskiest of the processes based on industry experience with requisite methods and tools. During reverse engineering, artifacts in the current operational environment are often in a state of flux. Much time can be lost administrating the on-going changes. Production of an obsolete product may result if the reengineering process does not coordinate maintenance and reengineering activities. During reverse engineering, there is a potential that many captured objects will be useless to the reengineering effort. The process and the tools may not be discriminating enough nor fast enough to handle the application volume.

Once captured, the application objects must be interpreted and analyzed for reuse in new systems. The captured objects must be analyzed to understand their use, context, and potential for reuse in other systems. The reengineering process should address partitioning, naming and defining the objects and place them in the larger context.

Many of the same risks associated with a typical software engineering project are present in the Forward Engineering risk area. Forward engineering should address the risks of adding additional requirements with an additional emphasis on integration. The addition of new requirements is a process similar to the maintenance release process and should consider typical risks associated with software engineering. The forward engineering process devotes considerable effort to integrating captured software objects with newly developed software, establishing detailed processes to deal with this perspective mitigates the risk of ineffective or poor quality system implementation.

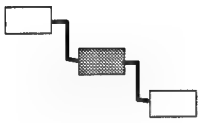
Table 2. Software Reengineering Process Risks

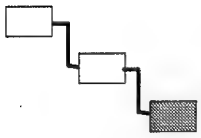
Process 	The software reengineering process can have a significant impact on the success of a reengineering effort. Not establishing, organizing, and executing the process effectively can expose the reengineering effort to considerable delays (false starts, redundant activities, etc.) and influence greatly the quality of the resulting product.	
Risk Area	**Impact/Exposure**	**Example Factors**
1 Preparation Activities 	Software reengineering often requires considerable preparation work establishing goals, analyzing applications portfolio, selecting reengineering strategy, determining cost/benefit, establishing support, developing plans, selecting technology, building the reengineering team, and acquiring resources. The absence of these preparation activities may cause the other reengineering activities to be inefficient or even fail.	
1.1 Establish Goals	Activities to identify requisite goals and obtain the commitment of management and staff are necessary for a reengineering project to be a success. In the absence of this activity there is considerable exposure to risks. The project risks becoming unfocused and inefficient without clear goals and management commitment to those goals. To the degree that objectives are measurable goals, objective metrics makes it impossible to substantiate claims of reengineering success or to document areas that contributed to a project failure. See the planning risks section to get a more detailed view of goals.	Short-term Goals • Determine scope of project • Get management commitment for resources (budget, staff, tools, etc.) • Establish responsibility and authority • Pilot reengineering approach Intermediate Goals • Evaluate captured artifacts for reuse • Plan forward engineering Long-range goals • Complete reengineering effort • Demonstrate cost/benefit
1.2 Analyze Application Portfolio	Activities to examine existing application software and data are necessary to get insight into the appropriate strategy or approach for reengineering. In the absence of this activity there is some exposure to missing important aspects of the product before establishing a reengineering plan. See Product risks for a more detailed view of application portfolio exposure.	

Risk Area	Impact/Exposure	Example Factors
1.2.1 Obtain Application	Activities to inventory application software and data descriptions are necessary to identify products for evaluation. Without a plan to identify and obtain the applications under investigation there is an exposure to not understanding the reengineering problem. Access to applications and tools to evaluate them is essential for successful analysis of the application portfolio.	Development artifacts • requirements, design, source code, test material, CM material, and installation material Operations artifacts • manuals and logs User documentation artifacts • reference manual, tutorials, and primers Data artifacts • schema descriptions, data item descriptions, and bridges
1.2.2 Evaluate Application	Activities to evaluate the applications software are important from both technical and functional standpoints. Verification of project goals as obtainable, planning of strategies, and calculation of costs and benefits are not possible without an evaluation of the quality and appropriateness of the product to be reengineered. See product risks section for more detail on these exposures.	Product usefulness • utility and duration of utility Maintainability • stability, reliability, understandability, size, complexity, traceability, and self descriptiveness Importance • to business process, organization, industry, and individuals Expected longevity
1.3 Select Reengineering Strategy	Activities to determine an appropriate reengineering strategy involves people from the business and technological arenas. Absence of this activity will impact the reengineering by missing important business and technical implications known by these groups. Not examining alternative approaches introduces a risk of the reengineering effort failing by not producing an improvement over the current situation or the effort may produce sub-optimal results. Details of the impacts for strategies are in the planning risks section.	Pilot Reengineering Approach Select Approach Plan for Large-Scale Implementation

Risk Area	Impact/Exposure	Example Factors
1.4 Analyze Cost and Benefit of Reengineering Effort	Activities to determine the cost and benefit of the reengineering activities are necessary to justify the project's start and to confirm its eventual effectiveness. Assessment of management support, development of informed business decisions, and eventual project success is not likely without such an evaluation.	Potential costs drivers • environmental, complexity, personnel, and applications familiarity Potential benefit drivers • wider reusability, reduced maintenance effort, consistent application use of process and data, and reliability
1.5 Establish Support for Reengineering Effort	Activities to enlist the support of the management, business users, and technical staff are necessary. Without the support of persons in each of these areas, the project may fail because of the lack of resources, domain expertise, or technical staff commitment.	Cost/Benefit Reviews Goal Reviews Technical Reviews Demonstrations and Pilot Projects History of similar projects
1.6 Develop Reengineering Plans	Activities to produce reengineering project plans are necessary to record important strategy, objective, schedule, resource, assumption, and approach information. The impact of not producing and getting a plan approved will be low management commitment and potentially disastrous implementation. Activities for evaluation at various points in the reengineering process are necessary to determine effectiveness of strategy and methods. In their absence, there is little control to prevent the economics of the effort from getting out of hand. Corrections to ensure goal attainment is not possible without these activities. Also see the planning risks section.	Strategy, objective, schedule, resource, assumption, and approach. Metrics Changes in strategy, tactics and methods Key Decision Points Product Evaluation Quality Program Principles

Risk Area	Impact/Exposure	Example Factors
1.7 Select Reengineering Technology	Activities to identify appropriate reengineering procedures, methods, and techniques are essential. Without effective, consistent, and understandable methods, the reengineering techniques will not be as effective and delays may occur. If reengineering tools are not available to manage complexity, to derive semantics from reengineered objects, and to generate physical objects, then project methods become cumbersome, tedious, and error-prone.	Process information to be captured (activity, data, state, event, etc.) Metadata (entities, attributes, domains, cardinality, business rules, etc.) Configuration Management Tool criteria based on methods Customer Referrals Tool Integration Tool Support Tool familiarity Cost per seat
1.8 Build Reengineering Team	As with most software endeavors, reengineering requires the right staff in the right mix. Impacts of not doing this can be significant in terms of time, costs, and quality of output. Absence of application domain, system, development/maintenance, or reengineering skills in a project can impact adversely a reengineering effort. Team players are essential in a reengineering project since it is the combination of these skills that make it successful. In the absence of having the right skills, an activity must be in place to acquire skilled staff for the effort. In the absence of having the right amount of skills, an activity must be in place to improve these skills. More detailed information on this exposure can be seen in the personnel risks section.	Identify reengineering staff • Domain experts • Systems experts • Reengineering experts Acquire reengineering staff • Get staff from within organization • Hire from outside • Train existing staff Establish training • Application domain • Program/system (development and maintenance) • Reengineering (method and tools)

Risk Area	Impact/Exposure	Example Factors
1.9 Manage Reengineering Acquisition	The absence of activities to select reliable, competent contractors puts the project at risk. The absence of a process to monitor actual contract performance against plan increases the risk of missing early warnings or making timely corrections taken when necessary. The wrong reengineering contract vehicle can increase the risk of a contractor's poor performance. Execution of a cost plus fixed fee contract is appropriate for some less standardized forms of reengineering. Where there is more experience and standards, fixed fee contracts are more appropriate. Without specific reengineering provisions for contractor selection, contractor monitoring, engineering environment, configuration management, quality standards, and the like, project cost, schedule, and quality are at risk.	Select sources issues • IV&V contractors • Independent testing • Customer references • Competitive selection Establish contracts vehicle issues • Reverse engineering - Cost plus • Reuse analysis - Cost Plus• • Reuse engineering - Cost Plus • Forward engineering - Fixed Fee Monitor reengineering contracts issues • Reengineering cost, schedule, technical performance
2 Reverse Engineering 	Reverse engineering is probably the reengineering technical activity that exhibits the most risk. Much of this risk is due to third generation computer languages not being designed to express abstract information necessary for requirements and design specification. Even the part of the program that can express useful semantics, "naming", is often substandard in legacy systems. Therefore with or without tools, it's difficult to capture much information beyond program design. Even with design, there is usually considerable risk unless there is a large human component with the appropriate skills (see personnel risk section). For more detail on reverse engineering exposures, see the technology risks section.	
2.1 Capture Objects	Capture activities should focus on identifying reusable objects. If much time is spent on manual capture of low level artifacts, there is an exposure to high cost and low benefit. More detail on this exposure can be seen in the technology risks section.	Manual versus automated activities Instituting change control and storage of captured objects Capture of objects is typically at the physical design level

Risk Area	Impact/Exposure	Example Factors
2.2 Analyze Objects for Reuse	Absence of activities to analyze the captured objects for reuse exposes the effort to significant work to adapt them in the reengineered system. That is, major portions of the application may require substantial work to reuse therefore making it more cost effective to not use them but instead to redevelop them. Not determining useful objects can lead to many redundant objects not being reused. Not validating objects with domain and systems experts can lead to redundant and ineffective components being used.	Reuse issues • Generalization (wide set of uses Abstraction(requirements versus code) • Adaptability (ease to change) • Parameterized (configure items) • Functionality
2.3 Prepare Objects for Reuse	Failure to prepare captured objects for reuse exposes the reengineering effort to the inefficiencies of custom modification for each reuse. Failure to document for reuse can lead to no reuse. Providing a source of standard reusable software objects enables development of integrated applications, without capturing such objects the risk of non-integrated applications and cost escalation increases.	Organizing validated objects for reuse Adapting objects for reuse Custom modification
3 Forward Engineering 	Beyond the normal software engineering risks, forward engineering increases risk only to the degree that preparation and reverse engineering activities expose it to risk. However, there are risk interactions with forward engineering that should be considered. First is the exposure to more risk in a reengineering effort when there are additional or new requirements during the reengineering effort. The second is that reuse of captured components has a certain overhead association with it which exposes the effort to integration risks.	

Risk Area	Impact/Exposure	Example Factors
3.1 Add New Requirements	Activities that integrate new requirements with those reverse engineered from the legacy system add risk commensurate with the difficulty of the new requirements. By not preparing for these requirements to be added to the existing structure, there is considerable risk in not recognizing their interactions and the requisite work involved. These risks resemble risks encountered when enhancements are made to a legacy system (Sherer 1990). Requirements risks are detailed in the product risks section.	Difficulty of a reengineering project increases when new requirements are added to existing captured requirements Interactions with existing requirements Working within the constraints of existing requirements and design Changing business environment during reengineering effort
3.2 Integrating Captured Objects to Construct New System	A shift from the standard systems engineering activities (moving from requirements to design) to integration of captured software artifacts with newly constructed components is the emphasis of much of the work in the reengineering context. This means integration issues such as system interfaces, component interconnection, testing strategies, and infrastructure become more pronounced exposures. There are also some risks in integrating the reengineered application into the surrounding business, functional, application, or technological environment.	Captured components integrated into new system functions and filling in the holes Enterprise model issues Interaction of new and existing business initiatives Test suites may need considerable rework Reoptimization
3.3 Reimplement System to Reengineer in the Future	Without encouraging reengineering principles in the current implementation effort, exposure to many of the same problems that plague current reengineering efforts will be present in the future. In the absence of naming conventions, programming style guides, formal specifications, and the like, the reverse engineering efforts of the future will suffer. Ineffective reimplementation can cause unfavorable comparisons with the current application's performance and convenience, causing loss of benefits by stalling customer acceptance.	Ineffective reimplementation • absence of: • Naming conventions • Programming style guides • Formal specifications • Documentation standards Not encouraging reengineering principles in the current implementation

Risk Area	Impact/Exposure	Example Factors
3.4 Migrate Existing Data	While gateways to existing data can ease data migration concerns, once the system has been reengineered, migration of data from the existing system to the new system should eventually occur to reduce the risk of a loss of the former system's utility. Expenditure of much manual effort keying or scanning, messaging, and storing existing data in the new system may result without providing data bridges for migration. The risk of not migrating the data is a delay in the use of the new system which may jeopardize user acceptance and result in project failure. If the old data is corrupt or obsolete there may be a cost exposure for its reclamation. (Normally this would be considered after forward engineering but for the purposes of risks, it is considered here.)	Manual versus automated migration strategy issues Cost/benefit of migration May not be cost-effective to reclaim data
3.5 Conduct Post-Reengineering Assessment	After completion of the preparation process, the reverse engineering and forward engineering processes should be executed and periodically reevaluated. A project review is conducted at the end of the project based on pre-established measure to determine success and document lessons learned. In the absence of conducting this activity, lessons learned in the project may be lost therefore exposing future reengineering efforts in the organization to repeat some of the same problems. By not examining reengineering effectiveness (meeting goals), there is little chance of justifying future reengineering efforts.	Reengineering lessons learned Future reengineering justification Capture reengineering cost/benefit experiences

3.3 Personnel Risk Category

Like most software engineering endeavors, personnel issues are critical in software reengineering efforts. Reengineering staff Availability, Knowledge (both experience and training), Motivation, and Teaming all play key roles in the success of a software reengineering project.

An effective software reengineering team consists of application domain expertise, program and system expertise, and finally reengineering expertise. Application domain expertise ensures that the reengineered software will perform according to the requirements of the

application domain. Program and system expertise expedites the software reverse engineering process by bringing knowledge about the location of important functions in the software. Understanding the software is essential when capturing software artifacts both from the completeness and consistency perspectives. Software reengineering expertise provides the necessary reengineering planning knowledge to successfully carry out the effort as well as the methods and tools knowledge to reverse engineer and forward engineer the software system.

Motivations for a software reengineering effort can influence greatly its potential for success. If the outcome of the reengineering effort involves increased efficiency of the systems reengineered, there is likely to be a perception that people employed to operate and maintain these systems may be laid off. The reengineering team may even feel that they are working themselves and others out of a job. Management of expectations of the reengineering project's consequences is essential. Additionally, software reengineering is often challenging work. Software reengineering projects are particularly subject to the effects of inadequate incentives or perceptions of ill-fate.

Building an effective team is essential for the success of a reengineering effort. As important as the right skill on a reengineering project is the right mix of these skills. Constructing a reengineering team with a large number of program and system staff from the old system increases the potential of the resulting product looking a great deal like the original product. However, if there is a large influence from the reengineering group and domain experts, there is a potential that the system may look radically different, even taking on substantial new functionality.

The balance of the reengineering team is important. The leadership should be from the program and system staff but should rely heavily on reengineering expertise to assist in system architecture decisions. The application domain experts should have a lesser role, more like an advisor to the functionality and testing. Not having the right staff availability or balance will present the same kinds of risk. For example, if program and system staff are not available during reverse engineering, there is considerable risk that there will be delays and that the artifacts captured will be less useful.

Although the software reengineering Personnel Risks category does not have a large number of risk areas, it is very important to the success of a software reengineering effort. We have split Personnel Risks out into a separate category because of the vital role it plays in a reengineering effort. By not paying requisite attention to personnel issues, reengineering efforts are subject to the hazards of poor reengineering planning, execution, costly rework, delays, and possibly failure. The risks identified in the personnel risks category impact the "Project Team" and the "Allocate Resources" activities as defined in the *CIM Software Systems Reengineering Process Model.* The following table outlines software reengineering personnel risks.

Table 3. Software Reengineering Personnel Risks

Personnel 	Like software engineering risks, software reengineering risks have an important component that involves personnel capability, availability, motivation, and teaming. The impact of not having available, knowledgeable, and motivated staff is a reengineering project that suffers low productivity, poor quality products, and potentially dissatisfaction with the resulting system.	
Risk Area	**Impact/Exposure**	**Example Factors**
1 Availability	Given that reengineering is a relatively new discipline and that there are frequent advances occurring in this technology, personnel apprised of the most effective reengineering techniques for the project are highly valued. This means that they are in demand and are often shared across multiple reengineering projects. The exposure here is that a planned reengineering resource may not be available enough to be effective therefore potentially delaying progress and/or producing substandard products from the reengineering effort. This situation is also true with domain experts and key systems staff. Lack of commitment by management to allocate requisite reengineering resources can also subvert the effectiveness of a software reengineering project.	Staff availability at the right time Spreading reengineering resources across too many projects Availability for requisite training Availability for planning
2 Knowledge - Experience/Training	The absence of knowledge of the application domain, the current system implementation, and/or reengineering project to considerable risk. Lack of key knowledge can lead to delays and poor quality products. Experience and training represent knowledge in this case.	

Risk Area	Impact/Exposure	Example Factors
2.1 Application Domain Expertise	The absence of application domain expertise in a reengineering project can lead to considerable exposure to errors in understanding the artifacts captured from the existing system. Without this knowledge, there may be considerable risk that newly reengineered software may not perform the functions necessary to support the business process. The potential impact here is not understanding the information technology requirements necessary to carry out the business process. Validating the reengineered system depends heavily on application domain expertise. Without it, the result may be considerable rework attempting to meet elusive system requirements. In effect, absence of application domain expertise leaves the reengineering project exposed to producing the wrong software for the job at hand.	Knowing functions supported in the business Business rule knowledge to support data model Knowledge of future uses of and advances in the business systems
2.2 Program/System Knowledge	The absence of program and system knowledge while reengineering a software system can lead to lengthy excursions through the current system artifacts looking for essential information to incorporate in the reengineered system. Lack of this knowledge can expose the reengineering effort to errors in design and specification. Not knowing the maintenance characteristics, architecture, infrastructure, and location of software artifacts, can lead to costly delays and potential errors reengineering the existing system.	Architecture knowledge of current and future systems Program structure understanding influences software artifact capture Configuration knowledge to find important parts of system during reengineering

Risk Area	Impact/Exposure	Example Factors
2.3 Reengineering Expertise	Planning and executing software reengineering projects requires considerable reengineering expertise. Without reengineering expertise, there is considerable exposure to poor reengineering effort estimates, misuse of the reengineering methods and tools, and poor execution of the reengineering effort. Ultimately, lack of this expertise on a reengineering project can result in costly delays and mistakes.	Software reengineering methods and tools experience Software reengineering costing Reverse engineering and forward engineering implications
3 Motivation	One of the deadliest risks of a software reengineering effort is not recognizing the impact of the software reengineering project on the organization as a whole. Perhaps the reengineered system will work more effectively, but there may be significant impacts on people in the process. Alignment of the motivations of the reengineering team and the organization precludes the reengineering effort from being subverted by outside interests. A particular situation worth mentioning is where the reengineering team is working itself out of a job because many of its members are part of the maintenance staff. In this case, there should be incentives in place so that success of the reengineering project will reward all concerned. Otherwise, the project is exposed to tactics designed to cause failure.	If the software reengineering outcome does not benefit personnel involved, may not be supported fully by team Personnel incentives must be in place for the reengineering effort to succeed Position of staff after the project can influence the reengineering effort

Risk Area	Impact/Exposure	Example Factors
4 Teaming	In most cases, a reengineering team must immerse itself in a project because there are so many relevant issues needing attention at once. Not only are there product issues, business process issues, technology issues and the like, these issues often have a high degree of interplay in the resulting system. If the right skills mix is not available for the particular phases of the reengineering project, there can be significant impact on schedules and quality of results. Building an appropriate team can influence greatly the success or failure of a software reengineering effort.	Right skills mix Balance of domain, system, and reengineering skills • Bias towards old architecture with mostly existing maintenance staff • Bias towards completely new system with mostly reengineering staff • Bias towards new functionality with mostly domain experts

3.4 PRODUCT RISK CATEGORY

Products used and produced by a reengineering project are determined by the reengineering strategy and approach selected. The characteristics of the information system being reengineered influences greatly the tools and techniques that are applied an types of products that are able to be produced. A reengineering project performing reverse engineering, inserting new requirements, and forward engineering a new information system, will use and produce a comprehensive class of products. In contrast, a redocumentation effort or a project limited to reverse engineering will use and produce a relatively restricted product set.

The components and characteristics of an information system influence the risk associated with reengineering. Reengineering products vary depending on the type of application software, data environment, or platform. Each reengineering project may require different reengineering tools and techniques and will have different associated risks. The risks identified in the product risk category impact the majority of the interfaces in the *CIM Software Systems Reengineering Process Model.*

An organization's technical infrastructure affects the products of reengineering by limiting the tools and techniques that can be applied. Hardware and system platform considerations are important reengineering issues and can have an extensive impact on the risk of a reengineering project. The configuration management capabilities of the technical infrastructure determine the risk associated with reengineering projects.

The use of standards, linkages to the business process model, the organization's data environment, and relative importance of an information system are issues that influence the

products and risks associated with reengineering. Software engineering and information management standards and methodologies can reduce the preparation time and improve the quality of a reengineering project. The use of a business process model provides guidance to information systems planning and can facilitate concentration of a reengineering project on key products.

Table 4. Software Reengineering Product Risks

Product

Characteristics of the product influence greatly the potential exposure to reengineering risks. The difference between the existing and target product characteristics is a primary driver in reengineering effort and costs. In the case where there are only a few relatively minor changes in the product there is likely to be less risk than where there are many substantial changes in the product. There are other implications that involve the product such as its importance to the organization and its support to the enterprise as a whole that induces risks by not reengineering.

Risk Area	Impact/Exposure	Example Factors
1 Application Characteristics	The components and characteristics of an application influence the risk associated with reengineering the application. Changing an application's software, data, or platform when reengineering may require the use of different reengineering tools and techniques and will impact associated risks.	
1.1 Software	The techniques and tools used during different system development life-cycle phases influences the reengineering products that are produced. The availability and quality of the software products impacts the risk of reengineering software.	

Risk Area	Impact/Exposure	Example Factors
1.1.1 Requirements	The format and quality of an application's software requirements documentation is a factor in determining the risk for a reengineering project. Requirements that are documented as informal text present a greater risk to a reengineering project than requirements that are formally specified. If requirements are undocumented and must be derived from existing code the risk of losing or misinterpreting the current requirements is magnified.	Informal Text Defined/Specified Derived Requirements Quality • Stability/Volatility • Completeness • Consistency • Testability • Traceability
1.1.2 Design	The accuracy and availability of software design material influence the risk of reengineering. If the software's design is understood it will reduce the risk of misinterpreting the existing design when reengineering.	Architecture/High-level • Batch versus Interactive operations • Integrated data architecture • Application/systems architecture • Databases interactions • Distribution of applications Program/Detail • Program structure • Documentation
1.1.3 Source Code	The complexity and reengineering risk associated with software source code is directly related to the type of language that is used. The type of language used to implement an application's software is a determinant of the level of effort that will be required when reengineering the software. Use of business-oriented languages tend to have a lower level of reengineering risk because of the relative availability and maturity of tools. The value of reengineering and the difficulty presented when reengineering is dependent on the maintainability of an application's source code. Code that is not maintainable is often a prime candidate for reengineering but reengineering such code presents inherent risk when performing reverse engineering .	Languages • Business • COBOL, 4GLs, RPG, PL/1 • Scientific/Engineering - FORTRAN, C/C++, assembler, CMS-2 • Software Engineering - Ada, C++, Modula, Pascal • Database languages Maintainability • Stability • Complexity • Size Portability Completeness Consistency Use of embedded interface languages • User Interface (X Windows) • Database Management Systems (SQL) • Reports Generators (4GL)

Risk Area	Impact/Exposure	Example Factors
1.1.4 Test Material (test case/data)	The availability of accurate test material is a determinant of risk associated with reengineering that is dependent on the approach that is used. Test materials may have to be reengineered with the code to ensure that the reengineered code meets the original requirements and specification.	• Unit level • String • System • Integration • Functional
1.1.5 Documentation	User, operational, training, and maintenance documentation is often required by a reengineering project. Frequently semantic information about software is contained in this documentation and is difficult to obtain elsewhere. The risk and difficulty of software reengineering is increased when documentation is unavailable, incomplete, or inaccurate.	User manuals Installation Operations manuals Training Maintenance Manuals Development Documentation • Requirements • Design • Program • Testing
1.2 Reuse Libraries	Population of a reuse library during a reengineering project can provide reusable objects to assist with the forward engineering effort. Objects should be analyzed and restructured to improve shareability before being offered for reuse through a library. If analysis and necessary restructuring is not performed on code captured during a reengineering project there is a risk that the code will not be able to be used by other projects and the effort expended to capture it will have been wasted.	Reusable Objects Applicability of objects to other information systems Repository Support

Risk Area	Impact/Exposure	Example Factors
1.3 COTS Products (Commercial - Off-The-Shelf)	Reengineering application software using COTS products present a unique set of issues and risks. The internal structures and documentation of COTS products are usually propriety and difficult to change. COTS products may change radically when a new version is released. The impact of a change of this type could impact cost and schedule. If the acquisition of COTS to replace application software functionality is employed, some development process risks are avoided, but other risks may be incurred. COTS acquisition sometimes tends to fit the organization to the COTS product, instead of the other way around. This may risk not getting important requirements implemented while having a considerable integration effort incorporating implemented requirements. Additionally, with COTS packages, technical and legal aspects of maintenance may be risky, or not even under the project's control. That is, every time a new release of the COTS product comes out, its added utility must be weighed against the effort necessary to install and integrate it into the other existing custom and COTS applications.	Domain Products • Financial Systems • Billing Systems • Human Resource Systems
2 Technical Infrastructure Implications	The technical infrastructure of an organization must be considered when evaluating the risk of reengineering. The types of technology employed, the use of standards, the age of system components, and availability of tools are reengineering risk determinants.	
2.1 Host/Target Platforms	Current host and target platforms are important issues when considering reengineering approaches and have an extensive impact on the risk of a reengineering project. If an information system is being reengineered to migrate to a different platform the risk of reengineering is increased.	Mainframe to client/server File management to DBMS 3GL to 4GL Migration to Object Oriented Operating System differences Connectivity between platforms

Risk Area	Impact/Exposure	Example Factors
2.2 Reengineering Platform	The availability of reengineering tools and techniques is dependent on the platform on which the reengineering is being performed, as well as, the host and target platforms. The availability and use of tools are important factors when considering the extent and risk of reengineering.	Reverse engineering tool support Forward engineering tool support Application generators Repository support I-CASE tools support Tool interoperability
2.3 Development Environment	The similarity of an organization's development environment to the host and target platforms can have an impact when reengineering an information system. If the development environment is different from the host or target platform, system interoperability, integration, and testing risks may be increased.	Applications Programming Interface Programming environment
2.4 Configuration Management	Configuration management and control products are valuable when pursuing reengineering. Configuration management tools help to ensure that the intended versions of software are being used by the reengineering effort and reduce the risk of reengineering.	Change Tracking/Control Configuration Control • CM Libraries • Build Scripts
2.5 Use of System Services	The availability and use of system services effects the risk of reengineering. The integration of an application's software with system service components must be considered when changing platforms.	File Management Data Management User Interface • Textual/command line • Graphic
3 Standards	Software engineering and information management standards and methodologies can reduce the preparation time and improve the quality of a reengineering project. The employment of naming, programming, and documentation standards influence the risk associated with reverse engineering a system. The adoption of methodologies and standards accommodates the implementation of reengineering techniques. Software engineering standards improve the quality of a reengineering project by providing mechanisms for developing consistent, structured, documented code.	Internal to organization • Programming • Development • Configuration Management • Documentation • Naming • COTS External to organization • Repository • Software Engineering Environment • Tool Standards

Risk Area	Impact/Exposure	Example Factors
4 Characteristics of Data	Data reengineering or restructuring is a major component of information system reengineering. The approach and techniques used when reengineering data are dependent on the level of abstraction (physical, logical, or conceptual) required to meet the selected reengineering objectives. As an information system is abstracted to higher levels, such as logical and conceptual, risk increases due to the amount of manual intervention normally required.	
4.1 Conceptual Model	A conceptual model is an information model that covers a wide business area and normally depicts major entities (or subject areas) and relationships. Reverse engineering data to a conceptual model has a high level of risk due to the distant relationship of conceptual entities to actual implemented data structures. A conceptual model reduces the risk of redundant data structures or processes by modeling a functional area's information requirements independent of technology considerations.	Reverse engineering of an information system's data structures to a conceptual model Integrate an information system with a wider scope of systems
4.2 Logical Model	Reverse engineering an information system's data structures to a logical data model is a reengineering activity that documents the data requirements of the current information system. Reverse engineering a logical data model entails a high-level of risk due to the lack of semantic information contained in data structures or code. The capability of tools to assist with the forward engineering of a logical model to a physical model can influence the risk if reengineering a system. The lack of a logical model for a system being forward engineered introduces a risk of the system being designed without a clear understanding of requirements.	Comprehensiveness of data requirements Number of data entities Number of attributes Number of relationships Use of "logical or business" names Use of standard data element names

Risk Area	Impact/Exposure	Example Factors
4.3 Physical Model	Reverse engineering a physical data model or database design for an information system is dependent on the implementation language. If a database management system is used, especially if supported by a data dictionary, the process is usually well supported by automated tools and risk exempt. Languages whose data definitions are not well documented or are not supported by automated reverse engineering tools are of a higher risk. The use of tools that generate database definition language statements reduces the risk of a design not being implemented as intended during the forward engineering process.	Completeness Number of tables/rows/ columns Number of primary and foreign keys Database design • Structure • Amount of tuning/optimizations
4.4 Access Methods	An information system's data access method can influence the reengineering process by adding complexity to the process of separating data from software processes. If a language's data access is tightly coupled to the platform or infrastructure a risk of substantial changes is incurred when changing the platform or infrastructure.	SQL Query By Example GUI products
4.5 Volume of data (for migration)	A system with a large amount of legacy data increases the risk of reengineering by preventing alternative data migration options (e.g., reentry of data) from being cost effective.	Number of records Number of database tables Number of recorded relationships

Risk Area	Impact/Exposure	Example Factors
5 Enterprise Model Implications	An enterprise model is a high-level model of an organization's information resources documented by an enterprise's data subject areas (entity types or classes), functions, processes, and their interrelationships. An enterprise model provides guidance to information systems planning and can facilitate reengineering by providing focus and key areas to consider. Detailed activity and data models are used during business process reengineering and are reconciled with the organization's enterprise model. If an organization does not have an enterprise model, a risk of not selecting appropriate information systems is present when performing reengineering. An enterprise model is a key factor when establishing an integrated, shared data environment and should be considered when planning for reengineering.	Data Architecture Process Architecture Applicability to business process • Dependency on changing process activities • Dependency on changing technology Application documents
6 Age	The age of a system to be reengineered is a critical determinant of potential risk exposure. Older systems typically are not well documented, use obsolete technology, and are difficult to maintain due to multiple patches to the software.	Legacy systems Migration systems
7 Intended Longevity	The expected longevity of a reengineered software system is an indication of the system's importance to the organization's operations. If a system is intended to be used for an extended period of time, it is a viable candidate for reengineering. Reengineering a system that is not expected to be of use in the future would introduce the risk of wasting valuable organization resources on unnecessary functionality.	Strategic importance Core business operation systems

Risk Area	Impact/Exposure	Example Factors
8 Importance to Organization and Operations	The relative importance of an information system to an organization's strategic direction or operational effectiveness is a factor when considering reengineering. If a system is of strategic importance or critical to an organization's operations, a conservative reengineering approach may be adopted to reduce overall risk to the organization. If a system has aged to a point of where it is of limited utility or if a systems intended longevity is not expected to be substantial, reengineering may not be a viable option.	Strategic importance Mission Critical relevance

3.5 Technology Risk Category

Software reengineering technology risks are among the most misunderstood of the risks detailed in this taxonomy. Since there is relatively little published experience in software reengineering most of the evidence for this section had to be obtained from experienced reengineering staff interviews. Much of the experience for software reengineering has been in the commercial sector where reengineering experience is considered propriety and treated as a commercial asset. In the case that the experience is successful, the company views the information as a competitive advantage. In the case that it is not so successful, the company views it as "airing dirty laundry." In both cases, the information is seldom disseminated.

In examining software reengineering technology risks, it was observed that there are a host of different techniques applied to existing systems, each having a level of risk associated with it. Although, the risks could not be quantified numerically, they can be discussed in relative terms. Figure 12 illustrates the idea of relative risk among reengineering techniques as well as their relative time of effort.

Redocumentation represents the least risk among the reengineering techniques. This is because redocumentation does not change the implementation or function and does not produce artifacts that are manipulated beyond placement in a report. Restructuring is the next one up on the relative risk scale. Restructuring does change the implementation but not any function. It is typically applied to simple transformations under control of the tool. Also, restructuring is the most mature of the techniques represented. Translation is much like restructuring in that it changes the implementation and typically not the function of a software system. Like the other reengineering techniques, the level of risk can vary widely with the situation. If a translation is from COBOL-74 to the updated COBOL-85 language, then the risk is relatively low. However, if the translation is from COBOL-85 to the Ada language, there is considerable exposure to risk.

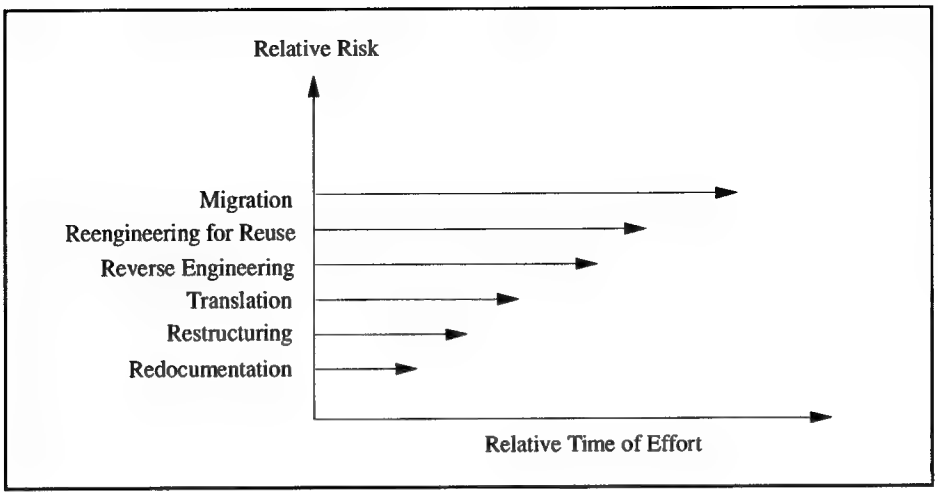


Figure 12. Relative Risk and Time for Reengineering Technology

Reverse engineering is next up on the scale. Although, reverse engineering does not change implementation or function, it does attempt to capture software artifacts and store them in a human malleable form. Since little more than program design information can be detected from source code, there is considerable risk attempting more.

Reengineering for reuse represents reverse engineering for the purposes of reuse. It does not necessarily capture higher levels of information (although it can), but it attempts to gather components to be reused later on other systems. Reengineering for reuse brings capture risks as well as selection, adaptation, classification, documentation, and storage risks. There are also external risks in using the reusable components once put in a library.

Reengineering consists of both reverse engineering and forward engineering. Unlike reuse engineering where the concentration is mostly on specific components, forward engineering often focuses on the entire system. Migration takes this a step further by reverse engineering functions from multiple systems and forward engineering them into one or more systems.

The risks identified in the technology risk category impact the "Identify Methodologies and Tools" activity, and also "Available Reengineering Technology", "Methodologies", and "Tools" as defined in the *CIM Software Systems Reengineering Process Model.*

In Table 5 below we examine software reengineering technology risks from the methods and tools perspective. First we examine methods described above with respect to their exposures and example factors. Then we detail tools risks in terms of capability, availability, and maturity.

Table 5. Software Reengineering Technology Risks

Technology

Technology can be effective in expediting a reengineering project and when relied upon inappropriately, a considerable risk exposure. In today's climate of making tools the answer to all situations, expectations surrounding reengineering tools need to be managed. If expectations are not managed, the impact on time, effort and costs can be devastating.

Risk Area	Impact/Exposure	Example Factors
1 Methods	Applying an inappropriate method to a reengineering situation can result in considerable waste. Risks vary largely between methods applied. Knowing when and how to apply a reengineering method is essential and when not aligned with the reengineering goals of an organization can expose a project to considerable risk.	
1.1 Redocumentation	Risks associated with the appropriate application of redocumentation technology are relatively low. Exposure comes when the documentation produced does not support the current development or maintenance process. If the redocumentation produces inaccurate data, this can lead to more exposure and concomitant risk since it can lead to mistakes in design and implementation decisions.	Redocumentation tools generate diagrams and metrics that may not be in the same representation as current documentation. Volumes of expensive documentation may be produced but not used.
1.2 Restructuring	Restructuring has the potential of obscuring the understandability of the transformed source code rather than making it more maintainable like it is designed to do. The impact here is that the software may be harder to maintain once it has been restructured. This implication warrants a thorough examination of the software before restructuring. The labor intensive nature of manual restructuring can represent considerable risk when automation is limited. Where translation software is available, this risk can be reduced a great deal. However, exposures still exist since restructuring tools seldom have the full range of language features covered for a given situation.	Language issues • Language covered • Multiple languages • Calls to system services Data structure issues • Physical design partially covered • Logical design not covered Complexity • Less complex but more difficult to change Performance characteristics • Increased size of code and image • May negate optimizations In code documentation may be rearranged

Risk Area	Impact/Exposure	Example Factors
1.3 Reverse Engineering	Reverse engineering technology is relatively immature but is increasing in capability as more attention is paid to its potential utility. Automated tools are still at a state where only parts of the physical design can be captured using tools. Although this saves considerable effort manually capturing the same information, exposures lie in the manual effort and utility of the information captured. The effectiveness of reverse engineering a software system can vary widely with the technology available (as well as the state of the product and the enterprise available to do the job (see other sections of taxonomy). One key issue that makes reverse engineering more risky than redocumentation is the intent that reverse engineered objects are placed in a form that can populate a repository and be manipulated.	Limited capture of business knowledge from source code • Languages not designed to express business rules • Evidence of business rules can be misleading Capture of physical design • Structure charts • Call graphs • Physical data elements Capture of logical design • Logical database design • User interface design • Communications architecture • Business rules Manual versus automated reverse engineering Evaluating captured objects for reuse • Recovered objects may not be useful or used
1.4 Forward Engineering	Forward reengineering exposures to risk are a function of reverse engineering success. Methods and tools may be inadequate to address forward engineering objectives therefore exposing the effort to costly rework. When captured artifacts can not be integrated with the existing structure, they cannot be shared across the reverse and forward engineering tools, resulting in extra work porting the information. Once completed, the reengineered system may not perform as anticipated leading to dissatisfaction from the users and costly optimizations.	Evaluating captured objects for reuse in forward engineering • Adaptability may be low • Usage potential may be low • Understandability may be low Program and variable labeling/ naming can be terse or misleading

Risk Area	Impact/Exposure	Example Factors
1.5 Translation	Risks associated with software system translation can vary widely with the type of translation and the extent of the effort. Exposures in translation come from lack of available translation software, poor quality code to translate (see product risks), major changes in architecture (e.g., centralized to distributed client/server), performance incompatibilities, and unanticipated translation obstacles. Like the other methods listed, impacts exist where substantial manual effort is needed to perform translation. Yet, automated tools often offer only limited assistance. Translation methods are typically as mature as the technologies that they involve; therefore when converting a system that employs older technology to newer technology the exposure also includes a product technology determinant (see product risks).	Language translation issues • Incompatibilities • Coverage Database translation issues • Structure incompatibilities • Data element naming Platform conversion issues • Operating system interfaces • System service coverage • Intertask communications Communications conversion issues • Protocols • Interfaces User Interface conversion issues • Type compatibility (GUI or CLI) • Mode of control (e.g., internal, external, or global)
1.6 Reengineering for Reuse	Capturing reusable artifacts from an existing system and populating a reuse repository can have exposure to risks. The act of capturing software artifacts can be difficult based on the type of artifact captured. In the absence of requirement and design specifications, evidence must be captured to derive requirements or construct a valid design which may take considerable resources to interpret and meet with limited success. Identifying and capturing reusable requirements, design, code, and test components is still a relatively new technology and may incur considerable risk. Once captured, the reusable components may need to be evaluated and adapted for reuse and then tested. This brings with it more exposure. Once the component has been placed in the reuse repository its identification and use can involve further exposure. Reuse of an artifact can expose the project where it is reused to any failures that the component may cause. This is multiplied across the number of places it is reused.	Reusable requirements • Hard to capture from low level artifacts • Entails risk to forward engineer Reusable design component issues • Difficult to capture from code • Redesign for reuse can be costly Reusable code component issues • Costs to reuse and maintain • Hard to understand how to use Reusable test component issues • Related to other artifacts • May need considerable modification to use with newly reengineered system May not get reused once captured Faulty components create exposure to failures proportional to utility of reuse

Risk Area	Impact/Exposure	Example Factors
1.7 Migration	Since migration frequently entails more than one of the other reengineering methods (often on multiple systems), it is likely to have considerable exposure to risks. There is often a strong dependency on business process reengineering efforts since migration systems typically have functions that support processes in multiple organizations. The consolidation of these functions into one or more systems have significant impacts on the reengineering effort. Identifying, capturing, and validating the software components that support requisite business functions requires considerable time and resources. Migrating one system to one other is the least risky. Migrating multiple systems to multiple other systems exposes the effort to very high risk. Once one or more systems have been reengineered into their target(s), their data must be harnessed, translated, and loaded onto the new system, therefore exposing the effort to further risks. Risks here include limited automated translation leading to costly manual effort, corrupted data, and redundant (and potentially inconsistent) data.	Migration of multiple systems into one or more systems Function allocation is a major concern for migration Business process dependency inherent in migration systems Data migration is a non-trivial task • Identifying data to migrate • Creating translation mechanisms • Removing redundancies • Correcting inconsistencies
2 Tools	Since tools sometimes automate major portions of reengineering methods, they can incur some of the same risks as described above. The most often seen exposure with reengineering tools is the dependence on tools that do not perform as advertised. Not only does this cause inefficiencies in completing reengineering tasks, the dialog with the vendor can be time consuming ultimately causing costly delays. Many software reengineering tools today are quite sophisticated and require considerable knowledge to make full use of them. The exposure here is that shortcuts may be taken in training that ultimately cost valuable time during their use.	Tools that do not perform as advertised Use of reengineering tools without requisite training Imbalance in utility-cost trade-off Appropriateness for the job

Risk Area	Impact/Exposure	Example Factors
2.1 Capabilities	Software reengineering tools provide capabilities to capture, translate, store, share, and manipulate software artifacts. However, not all reengineering tools do all of these and even those that do, do them to varying degrees. The exposure here is that a tool employed for reengineering may not have the requisite capability to assist in the reengineering effort, leaving much to be done manually. Even if the capability exists, it may not do it well; therefore exposing the project to delays and rework. Without a clear understanding of what the tools provide, the impact of misuse can be significant. To manage reengineering expectations, confidence in tool capability is important. Loss of confidence in tools can lead to lack of use and increased manual effort.	Capture issues • Identifying artifacts that do not exist (over-optimistic or in error) • Not finding requisite artifacts (incomplete or at wrong level) • Not organizing and storing objects Translation issues • Not able to translate to requisite form • Partial transformation • Transformation errors Language issues • Language not covered • System services interfaces not covered Repository issues • Doesn't use repository or share data Integration/Bridging issues • Tools are not integrated • No bridges to translate between tools
2.2 Availability	As with other tools, lack of reengineering tools in a reengineering effort can lead to reduced quality, performance, and slipped schedules. Reengineering tools are particularly susceptible to unavailability because many are relatively new and immature. Often, functions in these tools arrive late because features promised are in "the next release". The reengineering tool market is growing and aggressive techniques are being employed to capture that market. Without full reengineering tool availability information, there is exposure to mismanaging technology expectations.	Cost issues • High cost low benefit • Hidden costs not accounted for (e.g., installation, training, maintenance, etc.) Support issues • Limited assistance with installation • Inadequate training and/or schedules Platform issues (software or hardware) • Not available on platform • Limited functionality on platform Standard issues • Limited integration standards • Little or no reengineering standards

Risk Area	Impact/Exposure	Example Factors
2.3 Maturity	Reengineering technology for data is more mature than for software. Reengineering technology is mature for domains like management information systems and have relatively low risk while other domains like real-time systems have higher risks. Reengineering tools may be produced by new ventures looking to capitalize on a growing need for this technology and may lead to a tool being less capable than advertised. Without researching the stability of the tool vendor and its quality, a project may be exposed to unanticipated costly delays or failures when the vendor defaults or resulting from poor tool quality.	Area of application • Data issues • Software issues • Domain issues Vendor stability may be variable Quality of tool may be low

THIS PAGE INTENTIONALLY LEFT BLANK

SECTION 4

EXAMPLE USING THE TAXONOMY

During the latter part of Fiscal Year 1992, MITRE assisted the DISA/JIEO/CIM Software Systems Engineering Directorate and the U.S. Air Forces's weighted Airman Promotion System (WAPS) group with a pilot effort for exploring and demonstrating relevant reengineering technology. WAPS is representative of information systems employed by the DoD that will need to be migrated into the overall structure of DISA/JIEO/CIM information systems.

WAPS was a twenty year old COBOL system running on proprietary hardware. It was maintained by a frequently changing programmer base with widely varying styles of programming and development. WAPS was developed using little of the software engineering practices common in current software systems development.

This example will explore how the risk areas identified by the reengineering risks taxonomy applied to the WAPS reengineering project. Identification of reengineering issues and risks experienced during the WAPS reengineering project are presented using the format and risk areas of the reengineering risks taxonomy. The information used in this example was collected after the completion of the project and as such may provide a more detailed view of project risk issues than would typically be available to a project manager at project conception and initiation. The WAPS reengineering project experienced many of the issues and risks that are typical of converting existing legacy systems to new platforms using new technology. Table 6 identifies the characteristics of the existing and target system of the WAPS reengineering projects.

The interrelationship of the risk issues associated with reengineering are evident by examining the issues experienced by the WAPS project. Reengineering issues for all the major risk areas identified by the taxonomy were encountered during the WAPS project. The goals of the WAPS reengineering project drove the strategy, approach, technique, and tool selection for the project. The selected approach influenced the processes, personnel, products, and technology of the project. The data centered approach of the WAPS project placed primary importance on the need for data modeling skills, database management knowledge, and supporting tools. The technology used by the project team were determined by the identified goals, selected approach, and required processes.

It was recognized that WAPS is primarily a management information reporting system and the technical approach of this effort focused on the database aspects of the system. The key issues that drove the technical approach toward data reengineering were ineffective data management and the difficulty of WAPS software changes. The introduction of database technology enabled the WAPS development and maintenance staff to effectively manage their

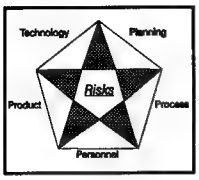
data and handle continuous changes to the system. The risks associated with the technology insertion and other related reengineering issues are documented by Table 7.

Table 6. Summary of WAPS Migration Characteristics

Current WAPS Characteristics	**Target WAPS Characteristics**
• Language: COBOL-74 125 programs, ~ 120 KSLOCs	=> Ada and 4GL
• Platform: Honeywell DPS-8000	=> UNIX-based AT&T 3B2
• Data management: "hard-coded"	=> ORACLE RDBMS
• Software documentation: very limited	=> DoD-STD-2167A
• Reasons for Modernization - Increasing maintenance costs - Inefficient manual processes - Dependent on aging platform - No integration	• Goals of WAPS Redesign Effort - Reduce maintenance costs - Automate manual procedures - Introduce new technology - Enable future integration

Table 7. WAPS Planning Risks Taxonomy Example

Planning



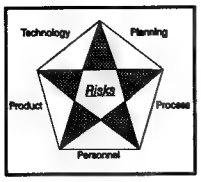
The WAPS reengineering project was executed as a complete redesign that pursued multiple disparate goals and objectives. The amount and nature of the change attempted by the project introduced a high degree of risk to the planning risk areas.

Risk Degree: High

Risk Area	Exposure	Mitigation/Impact
1 Reengineering Strategy	The WAPS project used what would be classified as an innovative or radical strategy for reengineering. The WAPS project's goals were based on the changing of the information system's data structures and platform. These changes necessitated the reengineering of the majority of the application's software processes as well .	The total redesign of the system and change of technology introduced a large amount of risk, especially in the areas of schedule and cost. One of the mechanisms for mitigating these risks was the plan to continue to use the existing system in parallel with the new system for a period of time.
2 Reengineering Approach	The reengineering approach used by the WAPS project team was one of a complete change to the implementation with minimal change to the supporting functions. This approach allowed the project to focus on technology issues without also incorporating substantial change in system functionality.	The selected approach provided necessary focus to the project but it prevented the implementation of functional improvements that were identified.
3 Reengineering Goals	The reengineering goals of the WAPS project were clearly identified to provide focus to the project and a method of precisely measuring the project's achievement. The goals of the WAPS redevelopment effort were to modernize the system and its infrastructure to improve responsiveness to user needs and to reduce current maintenance costs and improve system maintainability, reliability, and portability.	Well-defined goals for the project provided guidance to the technical approach and design decisions.

Table 7. WAPS Process Risks Taxonomy Example

Process	The software reengineering process used during the WAPS project was initially extensively a manual process and focused on data and database conversion issues. The reverse engineering and forward engineering activities were able to take advantage of CASE tools to automate part of the process. The process risk areas of the WAPS project were exposed to a medium to high level of risk due to the manual nature of the majority of the process and the relatively new technologies used. Risk Degree: Medium/High



Risk Area	**Exposure**	**Mitigation/Impact**
1 Preparation Activities	The time and effort required to establish plans and contracts, acquire necessary tools and training, and to resolve other management issues was sizeable. The project's goals, reengineering strategy, and approach were clearly defined and well established at the being of the project. This enabled the project to continue with a well-defined direction after initial start-up delays. Secondary issues such as analyzing the application portfolio, obtaining the source code for reengineering, and analyzing the application source code proved to be more labor intensive than anticipated.	The preparation activities of the WAPS project were numerous, with their impact not fully realized until the project was well underway. Preparation issues added pressure to the project's schedule and could have added substantial additional risk without the establishment of definitive project planning procedures. Without substantial input from domain experts the project could have been significantly delayed.

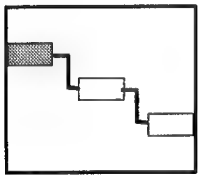


Table 7. WAPS Process Risks Taxonomy Example (Continued)

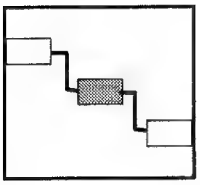
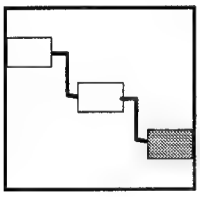
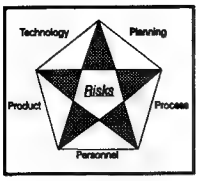
Risk Area	Exposure	Mitigation/Impact
2 Reverse Engineering 	Reverse engineering is an extensively manual process that introduces a large amount of risk. The missing and nonstandard documentation available for reverse engineering WAPS contributed to increasing the risk associated with the process. The reverse engineering phase of the process was strongly enhanced by the availability of an automated reverse engineering tool that captured identified data structures from the existing COBOL source code. The analysis of the captured objects and abstraction of the objects to a generalized information model was facilitated by having both information engineering and domain knowledge expertise.	WAPS would have been exposed to additional risk during reverse engineering had access to required domain knowledge not been available. The reverse engineering phase would have been exposed to considerable risk without a well-balanced project team and a robust reverse engineering tool.
3 Forward Engineering 	The forward engineering phase of the WAPS projects was exposed to minimal risk due to the limited amount of new requirements, use of sound software engineering principles, and adequate supporting technology. The only significant risk identified during the forward engineering phase of the project was due to the use of new tools and the systems move to a new platform.	Early identification of the associated platform and tool risks through use of a reengineering taxonomy would have helped to minimize risk exposure by the WAPS project. Tool risks were reduced by vendor training and knowledge transfer from the reengineering team's information engineers.

Table 7. WAPS Personnel Risks Taxonomy Example

Personnel



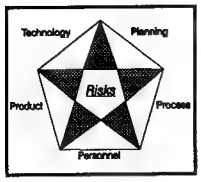
The main risk issues associated with personnel during a reengineering project: availability, knowledge, experience, training, motivation, and teaming; were mitigated by the selection and organization of the WAPS project team. This risk category was of a relatively low exposure to risk during the WAPS project.

Risk Degree: Low

Risk Area	Exposure	Mitigation/Impact
1 Availability	Information engineers, application domain experts, and system experts were available for the project.	The only substantial personnel risk experienced by the WAPS project was the geographic separation of project team members. This risk was mitigated by concentrated working group meetings and the use of electronic mail facilities.
2 Knowledge Experience/Training	The WAPS project team had domain knowledge experts, system experts, and information engineers who were trained in the methods and tools necessary for the project.	
3 Motivation	Members of the WAPS project team were enthusiastic and well motivated to meet project goals.	The WAPS project was not a threat to job security and did not attempt to implement radical changes to the organization.
4 Teaming	The WAPS reengineering projects team had a skill mix that provided all necessary elements for successful project implementation.	Periodic status briefings and project team meetings enabled the WAPS project team to stay abreast of project developments.

Table 7. WAPS Product Risks Taxonomy Example

Product



The WAPS main product related risks areas identified by the taxonomy were in the application characteristics, technical infrastructure, and standards areas. The WAPS reengineering project converted a medium sized legacy system to a new platform using minimal system services. The documentation was converted to a standard (DoD-2167A) format and the data architecture was significantly altered.

Risk Degree: High

Risk Area	Exposure	Mitigation/Impact
1 Application Characteristics	The existing WAPS application software requirements and design were not well documented. Inadequate documentation presented a risk of misinterpretation and schedule slippage due to the time spent attempting to understand the system. The application software source code was written in a business language (COBOL) that is relatively easy to understand and well supported by tools. However, the source code was several years old with added complexity and lacked stability due to frequent modification.	The lack of application documentation increased the project's schedule and the dependency on domain knowledge expertise.
2 Technical Infrastructure Implications	The WAPS technical infrastructure presented the highest degree of risk for the project. WAPS was migrated from a COBOL file management mainframe based system to a server based system using Ada, a fourth generation language, and a relational database management system. The number and types of technical infrastructure components being changed produced a high-degree of risk. The WAPS project reengineering tools, development environment, current system, and target system each operated on different platforms. The use of multiple platforms introduced transfer, conversion, configuration management, and performance related risks.	Additional methods, prototyping, and integration testing could have been pursued to test components of the target platform to reduce the associated risks. Early identification of technical infrastructure related risks would have assisted with development of an effective configuration management solution and could have influenced tool and technique selection.

Table 7. WAPS Product Risks Taxonomy Example (Continued)

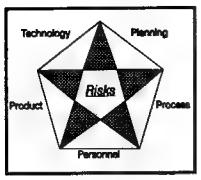
Risk Area	Exposure	Mitigation/Impact
3 Standards	WAPS was developed using the standards and techniques that existed at the time of inception. These older standards and techniques increased the project's risk during the reverse engineering of the system. The WAPS project team employed current software engineering techniques and standards during the forward engineering phase of the project.	The use of software engineering and information management standards and methodologies would have reduced the preparation time and improve the quality of a reengineering project. The employment of naming, programming, and documentation standards reduce the risk associated with reverse engineering a system. The use of current standards during the forward engineering process will increase the likelihood that the new system will exhibit reliability and stability that should reduce future maintenance costs. The use of standards also facilitated the capture of system components for future reuse and system integration.
4 Characteristics of Data	WAPS is a data intensive system that required extensive employment of data modeling and design techniques. The need to abstract a physical and logical data model from the existing COBOL file-based system required comprehensive use of reverse engineering tools and input from domain experts. Abstracting a design and requirements from the existing WAPS was a labor intensive effort that subjected the project to increased risk. The relatively large volume of data associated with WAPS placed additional requirements on the development of an effective data conversion plan.	The project team's data modeling and design experience and the availability of tools helped to mitigate the associated data design and conversion risks.

Table 7. WAPS Product Risks Taxonomy Example (Concluded)

Risk Area	Exposure	Mitigation/Impact
5 Enterprise Model Implications	The lack of an enterprise model and detailed functional models covering the information requirements of the WAPS introduced reengineering risks. Replacing the current WAPS with a new system using new technology without integrating the design into a detailed functional model and subsequently an overall enterprise model will impede the adoption of a shared data environment. The new system that replaces WAPS will still have the characteristics of a stovepipe system that is not integrated with a wider scope of information systems.	Not integrating the WAPS information model into a detailed functional model and the organization's enterprise model will present the potential for redundant processes and data structures in the future.
6 Age	WAPS was a twenty year old legacy system that had been modified extensively. The advanced age of WAPS exposed the project to additional potential risks.	WAPS age and lack of documentation required the presence of additional domain knowledge and the use of prototyping techniques to reduce risk exposure.
7 Intended Longevity	WAPS is expected to provide needed functionality to the U.S. Air Force for an extended period of time. This is an indication of the significance of the system to the organization's operations and indicated that it was a candidate to use reengineering technology to reduce the risk of project failure.	WAPS supports a core business operation which is needed to support the U.S. Air Force's personnel. Modeling and structured techniques were used by the project to ensure that a quality system was developed to support this required functionality.
8 Importance to Organization and Operations	WAPS supports the promotion function of the entire U.S. Air Force's non-commissioned officer staff. It is an important operational system that would risk the organization's capability to manage its personnel if its functionality was not available for an extended period of time.	WAPS is a system critical to the U.S. Air Force's personnel function. This dictated a somewhat conservative reengineering approach for the project.

Table 7. WAPS Technology Risks Taxonomy Example

Technology



The technology used by the WAPS reengineering project was determined by the overall goals of the project and the selected approach. The WAPS project was a reengineering effort that migrated the application to a new latform. The WAPS project was a data centered project that was relatively well supported by tools and therefore subject to somewhat less risk.

Risk Degree: Low/Medium

Risk Area	Exposure	Mitigation/Impact
1 Methods	The major risk with the reengineering technology used by the WAPS project was in the area of reverse engineering. The amount of manual intervention required by the reverse engineering phase of the WAPS project was not anticipated and introduced schedule and cost risks to the project.	The reverse engineering risks were mitigated during the WAPS project by having adequate availability of domain knowledge and information engineering expertise.
2 Tools	The data centered approach and the relatively well supported nature of the implementation language (COBOL) enabled reverse engineering tools to be effectively employed by the WAPS project to minimize risk exposure. The introduction of new tools and techniques during the WAPS project required that adequate training be included in the project's plan and budget. If training had not been included in the project plan the project would have been at risk to costly delays and improper tool use.	The availability of reverse and forward engineering tools for the forward engineering tools for the WAPS project reduced some of the reengineering risk exposure.

The impacts of the reengineering risks were evident during the WAPS project, especially in the area of preparation. The relatively new technology and techniques used during reengineering required considerable planning and preparation activities. The preparation phase of the WAPS project required significant effort that could have been identified by use of a risks taxonomy. Identification of the risks associated with the preparation activities would have allowed the project to plan for the activities and resources required.

The WAPS project would have benefited from employment of a reengineering risks taxonomy by having a clear view of the risks involved with the project. The key risk areas concerning planning and technology could have been identified earlier and included in the schedule. Identification and analysis of risk issues are key for mitigating associated risks and could have been profitably applied during the WAPS project.

SECTION 5

CONCLUSION

In this report we examined software reengineering risks in terms of planning for reengineering, the reengineering process, personnel involved in the reengineering effort, product considerations, and reengineering technology. Although this was by no means an exhaustive treatment of software reengineering risks, it is quite comprehensive. This taxonomy is designed to illuminate many issues critical to a software reengineering effort. It is intended to assist software reengineering program and project managers in identifying potential risks for a given situation.

This report summarized the reengineering risks problem; defined risk, software reengineering, and software maintenance terms; presented an overview of software reengineering; detailed the software reengineering risks taxonomy in tables for the five key risk categories (planning, process, personnel, product, and technology). It detailed these risk categories and associated risk areas in terms of potential exposures and impacts. An example use of the taxonomy was presented based on the Air Force's Weighted Airman Promotion System (WAPS) that was reengineering as part of fiscal year 1993 tasking.

DISA/JIEO/CIM is in a position to capitalize on the advancement of reengineering and to provide leadership in the field with the large number of systems that it plans to migrate. With this experience, there is a great potential to develop solid experiential models of software reengineering risks and costs. This taxonomy is a first step to embarking on that opportunity. One conclusion from this work is that there needs to be more publication of results in the reengineering field to validate software reengineering risk estimates and mitigation techniques.

The next step in this work is to validate the software reengineering risks taxonomy against AIS modernization projects. This will exercise the taxonomy and illuminate issues of its applicability and useability. Once the taxonomy has matured improvements and has been demonstrated to be effective at identifying software reengineering risks, it can be distributed to appropriate organizations for use by a wider audience.

LIST OF REFERENCES

Arnold, R.S., 1986, "Tutorial on Software Restructuring," *IEEE Computer Society Press Tutorial*, Catalogue Number EH0244-4.

Arnold, R.S. March 1990., "Heuristics for Salvaging Reusable Parts from Ada Source Code," *Software Productivity Consortium Technical Report* Ada_Reuse_Heuristics-90011-N.

Arnold, R.S., April 1991, "Risks of Reengineering," *Proceedings of Reverse Engineering Forum*, Washington University, St. Louis, MO.

Arnold, R.S., April 1992, "Common Risks of Reengineering," *IEEE Technical Committee on Software Engineering - Reverse Engineering Newsletter.*

Arnold, R.S., 1993, "Tutorial on Software Reengineering," *IEEE Computer Society Press Tutorial,* Order Number 3272-01.

Biggerstaff, T.J., July 1989, "Design Recovery for Maintenance and Reuse," *IEEE Computer.*

Boehm, B.W., 1989, "Software Risk Management," *IEEE Computer Society Press Tutorial,* Catalogue Number EH-0291-5, Pages 1-16.

Bohner, S., Fall 1992, "Software Maintenance: Sustaining Software Assets," *Information Systems Engineering Journal,* MITRE Publication.

Bush, E., *CASE for Existing Systems,* white paper, originally available from Language Technology, Inc., 27 Congress St., Salem, MA, 01970.

Charette, R.N., 1989, *Software Engineering Risk Analysis and Management,* McGraw-Hill Book Company, New York, NY, 1989.

Chikofsky, E.J., and J.H. Cross, January 1990, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software,* pp.13-17.

Chittister, C., R. Kirkpatrick, and R. Van Scoy, September 1992, "Risk Management in Practice," *American Programmer,* Vol.5, No.7, pp.30-35.

Davenport, T.H., 1993, *Process Innovation - Reengineering Work through Information Technology,* Harvard Business School Press, Boston, MA.

DISA/CIM, Reengineering Division, May 1993a, *Center for Information Management Information Systems Criteria for Applying Software Reengineering,* Defense Information Systems Agency, Joint Interoperability Engineering Organization, Center for Information Management, Software Systems Engineering Directorate, Arlington, VA, 22204-2199.

DISA/CIM, Reengineering Division, May 1993, *Center for Information Management Software Systems Reengineering Process Model - Version 1.0,* Defense Information Systems Agency, Joint Interoperability Engineering Organization, Center for Information Management, Software Systems Engineering Directorate, Arlington, VA, 22204-2199.

Functional Process Improvement: Functional Management Process for Implementing the Information Management Program of the Department of Defense, DoD 8020.1-M (Draft), Director of Defense Information, Office of the Secretary of Defense, August 1992.

Garnett, E.S. and J.A. Mariani, May 1990, "Software reclamation," *IEEE Software Engineering Journal,* pp.185-191.

Horowitz, B.M., July 1991, *The Importance of Software Architecture in DoD Software,* MITRE Publication M91-35, Bedford, MA.

Proceedings of the First Software Reengineering Workshop, Santa Barbara I, 21-25 September 1992, Department of Defense Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management.

Lehman, M.M. September 1980, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE,* Vol.68, No.9.

Lientz, B.P., and E.B. Swanson, 1980, *Software Maintenance Management,* Addison-Wesley, Reading, MA.

Medek, M. and H.G. Boylan, March 1992, *Data Integration and Migration Issues,* MITRE Technical Letter to DISA/CIM, TL-3910-92-008.

Risk Management, Concepts, and Guidance, March 1989, Defense Systems Management College publication, Fort Belvoir, VA.

Rochester, J.B. and David P. Douglas, October 1991, "Re-engineering Existing Systems," I/S *Analyzer,* United Communications Group, Vol. 29, No.10.

Rowe, W.D., *An Anatomy of Risk,* 1988, Robert E. Krieger Publishing Co., Malabar Fl.

Sherer, S., 1990, "Evaluation of Software Maintenance Risks," *Proceedings of the Conference on Software Maintenance.*

Sneed, H.M., July/August 1991, "Economics of Software Re-engineering," *Journal of Software Maintenance.*

Sneed, H.M., October 1991, "Bank Application Reengineering and Conversion at the Union Bank of Switzerland," *Proceeding of Conference on Software Maintenance,* pp.60-72.

Strassman, P.A., September 1992, "Shadow Warriors: Software Risk Management From the DoD Perspective," *American Programmer,* Vol. 5, No.7, pp.44-53.

System Safety Standard for Space and Missile Systems, August 1970, MIL-STD-1574, United States Department of Defense Military Standard 1574.

Ulrich, W., November/December 1990 and May/June 1991, "Re-engineering: Defining an Integrated Migration Framework", *CASE Trends,* four part series.

THIS PAGE INTENTIONALLY LEFT BLANK

INDEX